Fachhochschule Aachen
Campus Jülich

FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

Fachbereich: Medizintechnik und Technomathematik
Studiengang: Technomathematik

Quality of Service auf Infiniband

Masterarbeit von Matthias Nicolai
Jülich, den 24. September 2015

Die Arbeit wurde in Zusammenarbeit mit dem Jülich Supercomputing Centre, Forschungszentrum Jülich GmbH, angefertigt.

Diese Arbeit wurde betreut von:

1. Prüfer: Prof. Dr. Volker Sander

2. Prüfer: Dr. Dorian Krause

Diese Arbeit ist von mir selbständig angefertigt und verfasst. Es sind keine anderen als die angegebenen Quellen und Hilfsmittel benutzt worden.

Datum	Matthias Nicolai

Abstract

Im Zuge der Optimierung von Infiniband-Netzwerken, werden in dieser Arbeit die Auswirkungen eines Einsatzes von *Quality of Service* analysiert.
Dazu wird in die grundlegenden Theorien zu *Quality of Service* eingeführt und anhand dessen, die umgesetzte Architektur in Infiniband vorgestellt.
Aufgrund der unterschiedlichen Konfigurationsparameter werden die jeweiligen Auswirkungen auf Bandbreite und Latenz betrachtet. Zudem werden die Möglichkeiten zur automatisierten Trennung des Verkehrs in unterschiedliche Klassen untersucht.
Auf dieser Basis werden Konfigurationen für die spezifischen Anforderungen zweier typischer Anwendungsszenarien erarbeitet.

Inhaltsverzeichnis

1 Einleitung

Hinsichtlich der Planung und Konfiguration von Netzwerken nehmen Bandbreite und Latenz eine tragende Rolle ein. Ziel ist es, mit möglichst wenig Materialeinsatz, allen Anforderungen gerecht zu werden. Ein zentrales Problem, dass dabei gelöst werden muss, ist die richtige Dimensionierung der Verbindungen, sodass Überlastsituationen nur in den seltensten Fällen auftreten. Auch zukünftigen Anwendung muss dabei zu genüge gekommen werden. Sollte der Überlastfall dennoch eintreten oder existieren generell Anwendungen, die eine bestimmte Anteil der Netzkapazität benötigen, können Dienstgarantien definiert werden. Diese Dienstgarantien werden auch *Quality of Service (QoS)* genannt.
Im Jülich Supercomputing Centre des Forschungszentrums Jülich werden in verschiedenen performancekritischen Bereichen, wie dem Netzwerk eines Supercomputers, Infiniband-Netzwerke eingesetzt. Infiniband ist eine hochperformante Netzwerktechnik, die auf nachrichtenorientierten Punkt-zu-Punkt Verbindungen basiert. Gerade in den performancekritischen Bereichen haben Überlastfälle und die dadurch entstehende Verzögerungen im Betrieb besonders schwerwiegende Folgen.

Ziel dieser Arbeit ist es, die Frage inwieweit der Einsatz von *QoS* auf Infiniband im Umfeld des *Jülich Supercomputing Centre* empfehlenswert ist, zu beantworten.

Dafür muss sichergestellt werden, dass durch die Implementierung von *QoS* auf Infiniband alle erforderlichen Aspekte abgedeckt werden. Zudem müssen alle Auswirkungen, die der Einsatz von *QoS* hervorruft, betrachtet werden. Es muss folglich sichergestellt werden, dass die erforderlichen Konfigurationen einen konsistenten Zustand des Netzwerks erlauben, sowie die benötigte Performance geliefert wird.

In Kapitel 2 wird der Einsatz von *QoS* in Computernetzwerken motiviert. Darauf aufbauend werden die Grundlagen für zuverlässige Kommunikation in Netzen erarbeitet und die theoretischen Umsetzungen von *QoS* vorgestellt.
Die Realisierung von *QoS* auf Infiniband wird in Kapitel 3 behandelt. Dazu werden die angewendeten Kommunikations- und Kontrolltechniken vorgestellt. Neben der Betrachtung der *QoS-Architektur*, werden auch die Konfigurationsmöglichkeiten von *QoS* auf Infiniband erläutert.
In Kapitel 4 werden die durch *QoS* neu geschaffenen Möglichkeiten anhand zweier Anwendungsszenarien diskutiert. Auf dieser Basis werden Analysen zu Auswirkungen unterschiedlicher Konfigurationen auf Bandbreite und Latenz angefertigt.

Aus den Ergebnissen werden schließlich Konfigurationen für die diskutierten Anwendungsszenarien erarbeitet.
In Kapitel 5 werden die Ergebnisse dieser Arbeit zusammengefasst und ein Ausblick in darauf aufbauende Anwendungsmöglichkeiten gegeben.

2 Grundlagen einer zuverlässigen Kommunikation

2.1 Probleme in Computernetzwerken

Die Informatik befindet sich seit Jahren im Wandel. So geht der Trend immer mehr von sequentiellen Programmen in höheren Programmiersprachen, die auf separaten Rechnern laufen, hin zu parallelen Applikationen in verteilten Systemen. Im Zuge dessen wird auch vom *Paradigmenwechsel in der Informatik* gesprochen, durch den die Lösung komplexer Aufgaben in unterschiedlichsten Bereichen ermöglicht werden soll.
Durch die vermehrte Nutzung von verteilten Systemen, aber auch durch den rasanten Anstieg von internetfähigen Geräten, wächst der Datenverkehr im Internet in höhere Dimensionen. Einer Studie der Firma Cisco zufolge werde der IP-Datenverkehr im Jahre 2018 allein in Deutschland circa 3 Exabyte (3 Millionen Terabyte) pro Monat betragen. Davon würden in etwa 80 % auf Videostreaming entfallen [2].
Das Problem des hohen Datenaufkommens weitet sich demnach immer weiter aus und bildet eine große Herausforderung an die Infrastruktur von Netzwerken, vor allem an die des Internets.
Außerdem haben viele der neuen Technologien besondere Anforderungen an die Übertragung ihrer Daten. Die Internettelefonie beispielsweise benötigt eine garantierte maximale Verzögerung zwischen den Gesprächspartnern von 150 ms für eine sehr gutes Echtzeitempfinden. Bei einem Telefonat, bei dem die Verzögerung mehr als 400 ms beträgt, ist eine Unterhaltung nicht mehr möglich.[1]
Wird nicht nur telefoniert, sondern zusätzlich ein Livebild übertragen, wird von der Anwendung neben der geringen Latenz auch eine hohe Bandbreite benötigt, damit für die Teilnehmer einer Videokonferenz eine angemessene Qualität gewährleistet werden kann. Für das oben erwähnte Videostreaming wird außer der hohen Bandbreite vor allem eine gleichmäßige Übertragung der Daten benötigt, sodass dem Nutzer eine unterbrechungsfreie Wiedergabe zu ermöglicht werden kann.

Kurz zusammengefasst besteht ein Problem in der Bewältigung des anfallenden Verkehrs, sowie der Gewährleistung von spezifischen Parametern, wie Bandbreite und Latenz, einzelnen Verbindungen gegenüber. Diese Gewährleistungen oder auch Dienstgarantien werden, wie erwähnt, *Quality of Service (QoS)* genannt.

Zur Abstraktion des Problems von der Informationstechnologie, wird als Beispiel oft der Straßenverkehr in einer Großstadt herangezogen und dabei eine zentrale Straße betrachtet.
Die meiste Zeit des Tages ist das Verkehrsaufkommen in dieser Straße relativ gering. Doch morgens und abends zu Zeiten des Berufsverkehrs befinden sich deutlich mehr Fahrzeuge auf der Straße.
Technisch ausgedrückt erfüllt die Straße größtenteils ihren Zweck. Ihre Kapazität ist für den durchschnittlichen Verkehr groß genug. In Extremsituationen bricht jedoch der geregelte Fluss zusammen und es kommt zum Stau.
Nimmt die Anzahl der Überlastsituationen zu, muss entsprechend reagiert werden. Eine naheliegende Möglichkeit ist es, die Straße zu verbreitern, indem zusätzliche Fahrstreifen in jeder Richtung gebaut werden, obwohl diese größtenteils nicht benötigt werden. Dieser Ansatz wird *Overprovisioning* genannt.
Eine andere Möglichkeit ist es für bestimmte Fahrzeuge Privilegien einzuführen, wie etwa eine Bus- oder Taxispur. So wird gewährleistet, dass zumindest diese Fahrzeuggruppen trotz Staus zügig vorankommen können.
Zusätzlich sind im Straßenverkehr Regeln für besondere Situationen festgelegt. Kommt beispielsweise ein Polizei-, ein Feuerwehr- oder ein Krankenwagen mit Sirene oder Blaulicht, so haben diese Fahrzeuge absoluten Vorrang. Alle anderen Verkehrsteilnehmer müssen ihnen Platz machen, sodass sie auch im dichtesten Stau nicht behindert werden.

Das gleiche Prinzip lässt sich auch in paketvermittelnden Netzen anwenden. Dazu benötigt man, wie im Beispiel des Straßenverkehrs, unterschiedliche Spuren, sowie Strategien und Regeln für Ausnahmesituationen.

In Netzwerken müssen zudem andere, grundlegende Punkte gewährleistet werden. So wird von vielen Anwendungen eine zuverlässige und möglichst andauernde Verbindung benötigt. Die Aufgaben, die dazu von einem Protokoll der Transportschicht erledigt werden müssen, lassen sich in zwei Kategorien einteilen. Einerseits muss die Zuverlässigkeit garantiert und andererseits verbindungsorientierte Konzepte umgesetzt werden. Eine klare Trennung von Aufgaben in die Kategorien ist jedoch nicht immer problemlos möglich.
Zuverlässigkeit bedeutet in diesem Sinne, dass der Anwendung garantiert werden muss, dass alle Pakte in der richtigen Reihenfolge ankommen und keine, durch die Übertragung verursachten, Fehler enthalten. Dafür muss eine Buchhaltung, sowie eine explizite Bestätigung der Pakete durch den Empfänger implementiert werden.
Unter den verbindungsorientierten Konzepten werden die Regeln zum Auf- und Abbau von Verbindungen, sowie die der Überlastüberwachung, zusammengefasst. Auf Basis dieser Konzepte kann zusätzlich *QoS* implementiert werden. Darüber kann umfangreicher Einfluss auf die Auslastung eines Netzes, sowie die unterschiedliche Behandlung von Anwendungen genommen werden.

Im Folgenden wird zunächst die Überlastüberwachung am Beispiel des *Transmission Control Protocol (TCP)* betrachtet. Es ist das meist verwendete Protokoll zur Bereitstellung einer zuverlässigen Verbindung auf der durch das unzuverlässige *Internet Protocol (IP)* geschaffenen Basis. Dabei wird zwischen Fluss- und Staukontrolle unterschieden. Die Flusskontrolle dient zur Regulierung der Senderate aufgrund der Kapazität des Empfängers. Der genaue Ablauf wird in Abschnitt 2.2.1 beschrieben. Durch die Staukontrolle wird eine Anpassung der Senderate an die Möglichkeiten des Netzes ermöglicht. Die Implementierung in *TCP* wird in Abschnitt 2.2.2 behandelt. Beides sind Algorithmen, die in der Transportschicht des *OSI-Modells* anzusiedeln sind.

Darauf aufbauend werden in Kapitel 2.3 zusätzliche Möglichkeiten zur Anpassung des, von Verbindungen benutzen, Verkehrsprofils auf Ebene der Vermittlungsschicht vorgestellt. Auf Basis dieser Möglichkeit wird in Kapitel 2.4 *QoS* in Netzwerken auf Basis von *IP* eingeführt. Dabei wird die verbreitetste *TCP Version Reno* mit *IPv4* in kabelgebundenen Netzen betrachtet.

Als Grundlage für die Kapitel 2.2, 2.3 und 2.4 werden die Werke *Computernetzwerke* von *A. Tanenbaum* [20], sowie *Computernetzwerke - Der Top-Down Ansatz* von *J. Kurose* [14] verwendet. Alle weiteren Quellen sind entsprechend gekennzeichnet.

2.2 Überlastüberwachung

2.2.1 Flusskontrolle

Die Flusskontrolle wird dazu verwendet, die Senderate zu regulieren, sodass der Empfänger nicht überlastet wird. Die Regulierung wird aufgrund von Informationen des Empfängers und anhand der gemessenen Paketlaufzeiten im Netz vorgenommen.

Durch *TCP* wird zur Steuerung des Datenflusses ein Schiebefenster-Algorithmus verwendet. Bei der Flusskontrolle wird dabei vom *Advertised Window* gesprochen, da die vom Empfänger gewünschte Fenstergröße umgesetzt wird.
Zur Veranschaulichung wird im folgenden zunächst von einer festen Fenstergröße von n Bytes ausgegangen. Durch das Fenster wird angegeben, wie viele Daten gesendet werden dürfen, ohne dass eine Bestätigung eingegangen sein muss. Dies entspricht der Bandbreite, die durch den Sender maximal verwendet werden kann.
Wurden n Bytes versendet, wird gewartet, bis eine Bestätigung des Empfängers erhalten wurde. So lange werden keine weiteren Pakete versendet. Empfängt der Sender eine Bestätigung, wird das Fenster verschoben, sodass die bestätigten Daten nicht mehr im Bereich des Fensters liegen. Dadurch wird neuer Platz im Fenster geschaffen, wodurch weitere Daten gesendet werden können.
Im folgenden Beispiel sollen zehn Pakete verschickt werden. Zur einfacheren Darstellung wird von einer festen Fenstergröße, von drei Paketen, ausgegangen.
In Abbildung 2.1 ist die entsprechende Ausgangssituation dargestellt. Das *Advertised Window* ist *blau* und die noch nicht gesendeten Pakete *rot* dargestellt.

Abbildung 2.1: Ausgangssituation bei einem Schiebefenster-Algorithmus

Zu Beginn des Sendevorgangs wird das Fenster über die zu sendenden Pakete verschoben. Daraufhin werden die Pakete innerhalb des Fensters gesendet. Der Vorgang ist in Abbildung 2.2 dargestellt. Die gesendeten Pakete, für die noch keine Bestätigungen erhalten wurde, sind *gelb* dargestellt. Da kein weiterer Platz

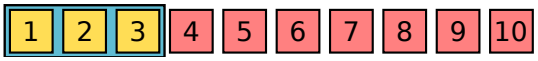

Abbildung 2.2: Warten auf eine Bestätigung

im Fenster ist werden keine weiteren Pakete gesendet. Es wird solange gewartet, bis eine Bestätigung durch den Empfänger erhalten wird.

Abbildung 2.3: Eingang einer Bestätigung

In Abbildung 2.3 ist der Eingang der Empfangsbestätigung des ersten Paketes *grün* gekennzeichnet. Danach wir das Fenster verschoben und es entsteht ein freier Platz im Fenster. Daraufhin wird, wie in Abbildung 2.4 dargestellt, das nächste Paket gesendet.

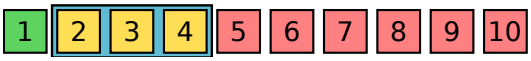

Abbildung 2.4: Verschiebung des *Sliding Windows*

Im Folgenden wird sukzessiv fortgefahren. Sollte eine Bestätigung eines Paketes ausbleiben, wird die Übertragung neu angestoßen. Dabei wird, je nach getroffener Vereinbarung beim Verbindungsaufbau, der *Go-Back-n-Algorithmus*, bei dem das gesamte Fenster neu übertragen wird, oder der *Selective-Repeat-Algorithmus*, der einzelne Pakete neu überträgt, verwendet.

Durch dieses Prinzip wird die Senderate dynamisch der Leistung des Netzes angepasst, da nach der initialen Übertragung des Fensters, nur neue Pakete verschickt werden, nachdem eine Bestätigung für ein bereits übertragenes Paket eingegangen ist. In einem langsamen Netz wird mehr Zeit benötigt, bis diese Bestätigung beim Sender eintrifft. Daher ist auch die induzierte Rate geringer. Entsprechend wird sich die Senderate erhöhen, wenn durch das Netz bessere Paketlaufzeiten ermöglicht werden. Die Zeit, die ein Paket vom Sender bis zum Empfänger und wieder zurück benötigt, wird *Round Trip Time (RTT)* genannt.

Damit durch den Empfänger Einfluss auf die Senderate genommen werden kann, wird die Größe des *Advertised Windows* variabel gehalten. Bei jeder Bestätigung eines Paketes wird die gewünschte Fenstergröße im Header mitgeschickt. Die Größe richtet sich oft nach dem zur Verfügung stehenden Empfangspuffer, kann aber auch durch das Betriebssystem gesteuert werden. Der Ablauf nach dem Verbindungsaufbau ist schematisch in der Abbildung 2.5 dargestellt.
Nachdem im Beispiel von einer Anwendung 2 KB in den Sendepuffer geschrieben wurden, werden diese versendet. Die Sequenznummer *(SEQ)* starten hier der Einfachheit halber bei 0. Aus Sicherheitsgründen wird durch *TCP* an dieser Stelle eine Zufallszahl verwendet.
Wurden die Daten empfangen, wird die korrekte Übertragung der Pakete mit der Bestätigungsnummer *(ACK)* 2048 durch den Empfänger quittiert. Die Zahl ergibt sich aus der Summe der vorherigen Sequenznummer und der Größe der empfangenen Nutzdaten in Bytes. Im gleichen Paket wird auch eine neue Fens-

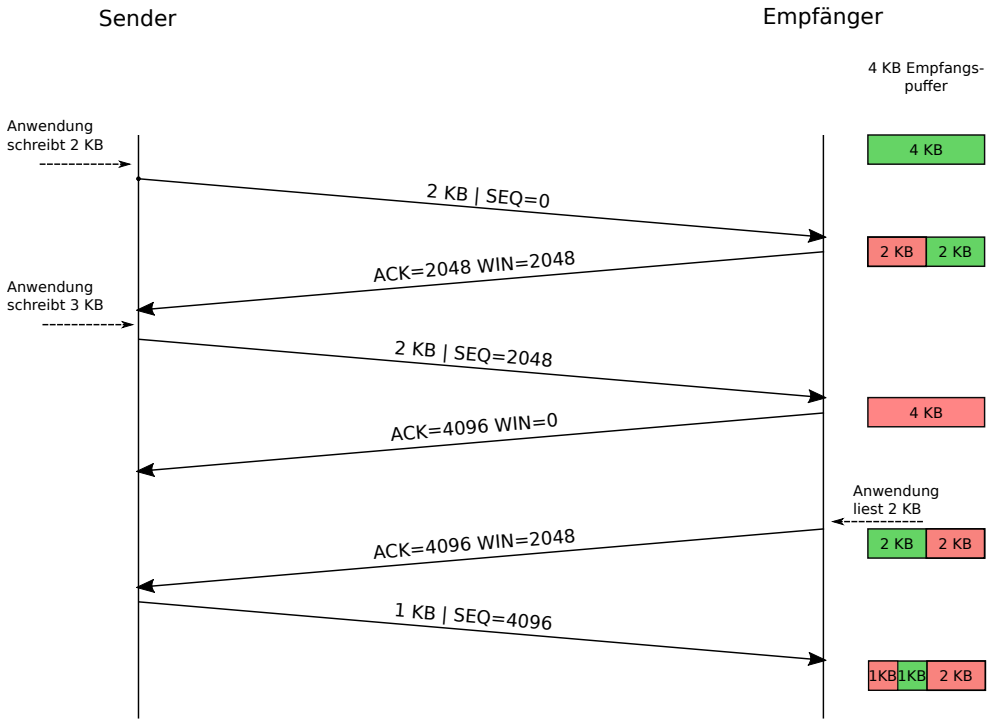

Abbildung 2.5: Veränderung der Größe des *Advertised Windows* [20, S. 642, Abb. 6.40]

tergröße *(WIN)* übertragen. In diesem Fall sind es 2 KB, da bereits 2 der 4 KB des Empfangspuffers durch die zuvor gesendeten Daten belegt werden.
Wurde die Bestätigung durch den Sender empfangen, werden weitere 3 KB in den Sendepuffer geschrieben. Von den 3 KB im Puffer dürfen aufgrund der Fenstergröße nur 2 KB versendet werden. Dafür wird die *SEQ* 2048 verwendet, da die Bytes 0 bis 2047 schon verschickt worden sind.
Nachdem die 2 KB empfangen wurden, ist der Empfangspuffer komplett gefüllt. Daher wird mit der Bestätigung eine neue Fenstergröße von 0 Bytes mitgeteilt.
Wurden Daten aus dem Empfangspuffer von der empfangenden Anwendung gelesen, wird darin Kapazität frei. Daraufhin wird eine neue Bestätigung mit einer Fenstergröße von 2 KB verschickt. Anschließend werden die restlichen Daten durch den Sender übermittelt.

Ein Problem des *Sliding-Windows* ist, dass im *TCP-Header* lediglich 16 Bit für die Angabe der Fenstergröße vorgesehen sind. Das entspricht einer maximalen Größe von 64 KB. In der heutigen Zeit wird dadurch die Kapazität der Übertragungsmedien bei weitem nicht mehr ausgereizt. Zur Lösung des Problems wurde die Option der *Fensterskalierung (Window Scale)* in den optionalen Bereich des TCP-Headers aufgenommen. Die Informationen zur Fensterskalierung umfassen 3 Bytes. Neben den formalen ersten beiden Bytes, wird im letzten eine Binärzahl

angegeben, die den Exponenten einer Zweierpotenz angibt, mit der die eigentliche Fenstergröße multipliziert wird. Aufgrund der Länge der Sequenznummern von 31 Bit, wird ein Maximum von 14 festgelegt, sodass keine Wiederholungen von Sequenznummern innerhalb eines Fensters möglich sind. Dadurch können Fenstergrößen von bis zu 2^{30} Byte (entspricht 1024 MB) verwendet werden. Die *Window-Scale-Option* wird einmalig bei Verbindungsaufbau zwischen Sender und Empfänger ausgehandelt und wird für die gesamte Dauer der Verbindung beibehalten.[7]

2.2.2 Staukontrolle

Bisher wurde durch das *Sliding Window* die Senderate an die Leistung des Empfängers angepasst. Zwar spielt dabei auch der Netzzustand in Form der *RTT* eine Rolle, auf die Größe des Fensters wird dabei allerdings keinen Einfluss genommen. Das hat zur Folge, dass bei Paketverlusten durch Überlast im Netz, mit gleichbleibender Fenstergröße übertragen wird. Dadurch wird die Situation noch weiter verschärft, wodurch noch mehr Pakete verworfen werden. In diesem Zusammenhang wird auch von einem *Congestion Collapse* gesprochen.
Zur Steuerung der Senderate in Abhängigkeit zur Last im Netz dient die *Staukontrolle*. Dabei gibt es zwei unterschiedliche Ansätze. Einerseits eine *Ende-zu-Ende Staukontrolle*, andererseits eine *Staukontrolle mit Unterstützung des Netzwerks*.
Bei einer *Ende-zu-Ende Staukontrolle* wird davon ausgegangen, dass bei Paketverlust oder einer im Vergleich sehr hohen Latenz, eine Überlastsituation im Netz vorliegt. Bei einer *Staukontrolle mit Unterstützung des Netzwerks* wird hingegen der Sender direkt von den Routern über eine Überlastsituation benachrichtigt. Dies kann entweder explizit durch ein zusätzliches Paket vom überlasteten Router an den Sender geschehen oder durch das Setzen eines *Header-Flags*. Der *Flag* wird durch die Bestätigung des Pakets durch den Empfänger an den Sender weitergeleitet.

Durch *TCP* wird eine *Ende-zu-Ende Staukontrolle* implementiert. Dies ist möglich, da die Übertragungsqualität in den heutigen Netzen so gut ist, dass davon ausgegangen werden kann, dass alle Pakete korrekt übertragen werden. Daher wird bei einem Paketverlust oder auch hoher Latenz direkt auf eine Überlastsituation geschlossen.
Dazu wird senderseitig ein weiteres Übertragungsfenster, dass sogenannte *Congestion Window*, initialisiert. Es wird im Grunde die gleiche Funktionalität, die das *Advertised Window* bietet, erneut implementiert.
Die Fenstergröße die tatsächlich beim Versenden verwendet wird, ist das Minimum des *Congestion* und des *Advertised Windows*.

Die Größe des *Congestion Windows* wird aufgrund unterschiedlicher Situationen angepasst. Wird ein Fehlerfall in einer Verbindung erkannt, wird ein Schwellwert

in Bytes auf die Hälfte der Größe des *Congestion Windows* gesetzt. Erreicht die Größe des Fensters den Wert, wird zwischen multiplikativem und linearem Wachstum des Fensters gewechselt. Dadurch soll die Senderate möglichst genau an den bestmöglichen Wert angenähert werden. Dieser Schwellwert wird in jedem neuen Fehlerfall auf die Hälfte der Größe des dann aktuellen *Congestion Windows* gesetzt.
Zu Beginn der Verbindung wird ein sogenannter *Slow Start* ausgeführt. Es wird mit einer geringen Senderate begonnen, die nach und nach gesteigert wird. Dazu wird das *Congestion Window* zunächst auf 1 gesetzt. Mit jeder eingehenden Bestätigung eines Paketes wird das Fenster um 1 vergrößert. Wird also ein gesamtes Fenster bestätigt, wird dessen Größe verdoppelt. Daher handelt es sich dabei um ein multiplikatives Wachstum. Erst in einem Fehlerfall wird von dieser Strategie abgewichen.

Auf Senderseite werden Fehler auf zwei unterschiedliche Arten erkannt. Entweder wird ein sogenanntes *Tripel-Duplikat-Acknowledgement (TDA)* empfangen oder ein *Timeout* eines Paketes festgestellt.
Bei einem *TDA* wird durch den Empfänger zum dritten Mal die gleiche Sequenznummer bestätigt. Dieser Effekt tritt auf, wenn Pakete korrekt empfangen werden, über die Sequenznummern jedoch empfängerseitig eine Lücke festgestellt wird. Dieses Paket wird über die Sequenznummer erneut angefordert. Auf Senderseite kann darauf geschlossen werden, dass nur ein einzelnes oder wenige Pakete verloren gegangen sind, da weitere Pakete empfangen werden. Daher wird das *Congestion Window* halbiert und der Schwellwert auf eben diesen Wert gesetzt. Es wird also direkt mit einer linearen Steigerung fortgefahren.
Wird durch den Sender ein *Timeout* festgestellt, das heißt, dass innerhalb einer bestimmten Zeit keine weitere Bestätigung eingegangen ist, wird dies als besonders schwerer Fehler in der Verbindung angesehen. Es wird davon ausgegangen, dass höchstwahrscheinlich auch die danach gesendeten Pakete nicht ankommen werden. Der Schwellwert wird auf die Hälfte des aktuellen *Congestion Windows* gesetzt und ein neuer *Slow Start* mit einer Fenstergröße von 1 ausgelöst.

In Abbildung 2.6 wird schematisch ein möglicher Verlauf einer Verbindung mit *TCP* nach dem Verbindungsaufbau dargestellt. Dabei wird der typische *TCP-Sägezahn* sichtbar. Zunächst wird mit einem *Slow Start* begonnen. Zum Zeitpunkt t_1 wird ein *Timeout* festgestellt. Daher wird der Schwellwert auf S_1 gesetzt und ein neuer *Slow Start* ausgeführt. Ab einer Größe von S_1 wird die das *Congestion Window* nur noch linear vergrößert.
An den Stellen t_2 und t_3 wird jeweils ein *TDA* empfangen. Der neue Schwellwert wird in beiden Fällen auf die Hälfte der jeweiligen aktuellen Größe des *Congestion Windows* gesetzt. Im Beispiel beträgt der Wert in beiden Fällen zufälligerweise S_2. Die Größe des neuen *Congestion Windows* wird ebenfalls auf S_2 gesetzt und jeweils direkt mit einem linearen Wachstum fortgefahren.

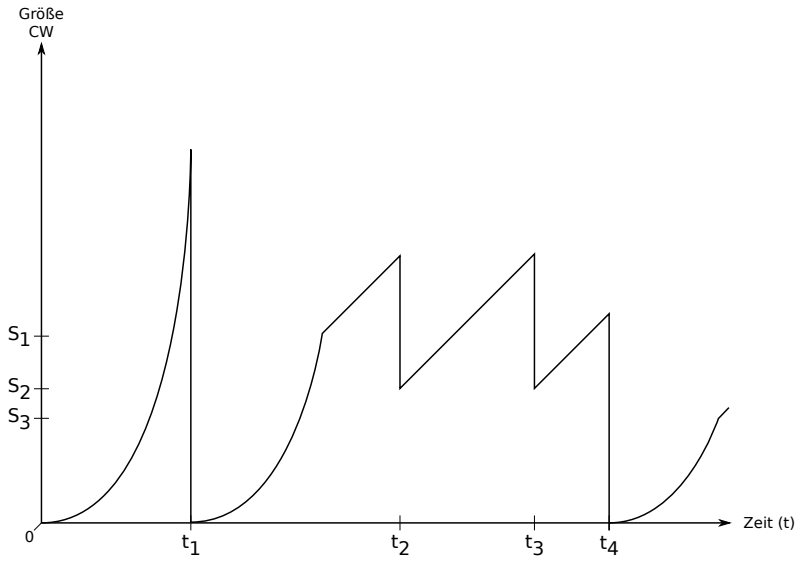

Abbildung 2.6: Verlauf der Größe des *Congestion Windows*

Zum Zeitpunkt t_4 wird ein weiterer *Timeout* festgestellt. Der neue Schwellwert wird auf S_3 gesetzt und ein neuer *Slow Start* ausgelöst. Ab dem Schwellwert wird wieder auf das lineare Wachstum gewechselt.

Durch das Zusammenspiel der *Stau-* und *Flusskontrolle* wird die Senderate in Überlastfällen sowohl beim Empfänger, als auch im Netz, reduziert. Effizienter wäre jedoch eine gleichmäßigere Belastung des Netzes. Dazu muss die Senderate genauer kontrolliert werden. Im folgenden Kapitel 2.3 werden dazu zwei Algorithmen vorgestellt, durch die auch im Bezug auf *QoS* eine zentrale Rolle eingenommen wird.

2.3 Traffic Shaping

2.3.1 Leaky-Bucket

Durch das Prinzip des *Leaky-Bucket-Algorithmus* wird eine simple und zugleich sehr effektive Herangehensweise geschaffen um die Staugefahr in einem Netz deutlich zu reduzieren. Als zentrale Ansatzpunkte werden die Vermeidung von *Bursts* und eine maximale Senderate herangezogen.
Bildhaft wird der Algorithmus durch einen Eimer mit einem kleinen Loch an der Unterseite dargestellt. Wird der Eimer unregelmäßig mit Wasser gefüllt, so bleibt die Abflussrate ξ, bis der Eimer leer ist, konstant. Wird die Kapazität σ des Eimers überschritten, so läuft das Wasser über den Rand des Eimers und geht verloren.
Übertragen auf das Versenden von Paketen in einem Netzwerk bedeutet dies folgendes: Liegen Pakete zum Versenden in einer Warteschlange mit der Kapazität σ werden sie maximal mit der Rate ξ auf die Leitung induziert. Wird die Kapazität der Warteschlange überschritten, so werden die Pakete verworfen. Die

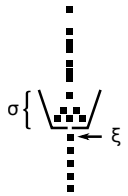

Abbildung 2.7: Funktionsweise des *Leaky-Buckets*

Funktionsweise des *Leaky-Buckets* ist schematisch in Abbildung 2.7 dargestellt. Die ankommenden Pakete haben einen zufälligen Abstand zueinander und werden im *Bucket* gesammelt. Auf der Ausgangsseite wird die Rate entsprechend der Konfiguration angepasst. Alle ankommenden Bursts werden somit ausgeglichen. Die Rate ξ kann generell frei gewählt werden. Sinnvollerweise sollte sie kleiner als die angeschlossene Bandbreite sein. Zur Verbesserung der Flexibilität wird die Einheit von ξ in Byte pro Sekunde anstatt in Pakete pro Sekunde definiert.

Für eine Implementierung der Warteschlange der Pakete kann ein endlicher Speicherbereich verwendet werden. Ist die Warteschlange komplett belegt, werden neu ankommende Pakete verworfen. Zusätzlich wird in jedem Zeittakt ein Zähler initialisiert, der die maximale Anzahl an Bytes enthält, die in einem Takt gesendet werden dürfen. Werden Pakete versendet, wird jeweils ihre Größe von diesem Zähler abgezogen. Ist das nächste Paket größer als der Wert des Zählers, wird in diesem Zeittakt kein Paket mehr gesendet.

Zur Verdeutlichung des Nutzens eines *Leaky-Buckets* ist in Abbildung 2.8 jeweils ein beispielhaftes Ein- und Ausgangsprofil dargestellt. Dabei wird angenommen,

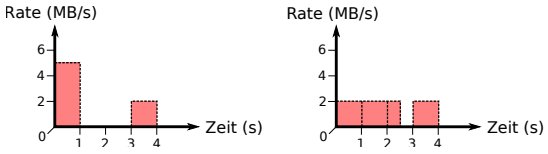

Abbildung 2.8: Ein- und Ausgangsprofil beim *Leaky-Bucket-Algorithmus*

dass die Kapazität der Warteschlange ausreichend groß ist, sodass keine Pakete verworfen werden. Die Abflussrate wird auf 2 MB/s festgelegt.
Auf der Eingangsseite erfolgt zunächst ein Burst von 5 MB/s für eine Sekunde. Nach einer Pause von zwei Sekunde gehen für eine weitere Sekunde Daten mit einer Rate von 2 MB/s ein. Durch den *Leaky-Bucket* wird der gesamte Burst aufgefangen und kontinuierlich mit 2 MB/s die gesammelten Daten weitergeleitet. Nach 2,5 Sekunden sind alle Daten versendet worden. Die nach der Pause gesendeten Daten werden nicht verzögert, da die Eingangsrate der maximalen Senderate entspricht.

Zusammenfassend bietet der *Leaky-Bucket-Algorithmus* eine effektive Möglichkeit die Nutzung eines Mediums einerseits zu begrenzen, andererseits eine gleichmäßigere Belastung zu ermöglichen.

2.3.2 Token-Bucket

Eine funktionelle Erweiterung des *Leaky-Buckets* wird durch den *Token-Bucket-Algorithmus* realisiert. Dadurch werden zusätzlich zur Beschränkung der Senderate Bursts einer definierbaren Größe zugelassen.
Bildhaft wird dazu wiederum ein Eimer definiert, in dem nun Tokens gesammelt werden. Diese Tokens werden beim Versenden von Daten verbraucht. Ist kein Token im Eimer vorhanden, kann auch kein Paket versendet werden. Durch die Kapazität σ wird daher die Anzahl der Pakete, die gleichzeitig versendet werden können, limitiert. Ein möglicher Burst wird somit begrenzt, allerdings nicht komplett unterdrückt. Der Eimer wird mit einer konstanten Rate ξ mit Tokens gefüllt. Diese Rate entspricht der maximalen Senderate, mit der nach einem Burst Pakete verschickt werden können. Ist der Eimer vollständig gefüllt werden die überschüssigen Tokens verworfen. Anders als beim *Leaky-Bucket* wird die Annahme wartende Pakete keinesfalls verweigert.
Die Funktionsweise ist in Abbildung 2.9 schematisch dargestellt. Der *Token-Bucket* wird mit konstanter Rate ξ von oben mit Tokens gefüllt, bis die maximale Kapazität σ erreicht ist. Wartet ein Paket auf Weiterleitung, wird soweit vorhanden ein Token entnommen und ein Paket versendet.

Die Implementierung kann über einen Zähler, der pro Zeitschritt um eine Konstante c erhöht wird, erfolgen. Dies entspricht genau der Rate ξ. Der Zähler wird

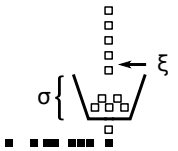

Abbildung 2.9: Funktionsweise des *Token-Buckets*

nur erhöht, bis er seinen maximalen Wert σ erreicht hat. Wird ein Paket gesendet, wird der Zähler um eins reduziert. Alternativ kann, wie auch beim *Leaky-Bucket*, das Prinzip in Bytes und nicht in Paketen umgesetzt werden. Dazu wird der Zähler in jedem Zeitschritt um das Produkt aus c und einem Faktor k erhöht. Der Faktor k ist eine Konstante in Bytes. Beim Versenden eines Paketes wird der Zähler entsprechend um die Länge des Paketes reduziert.

Zur Verdeutlichung des Sendeablaufes mit einem *Token-Bucket* ist in Abbildung 2.10 ein Ein- und Ausgangsprofil schematisch dargestellt. Der Bucket hat eine Kapazität von 2 MB und wird mit einer Rate von 1 MB/s gefüllt. Im Beispiel wird angenommen, dass der Bucket zu Beginn voll gefüllt ist. Der eingehende Burst

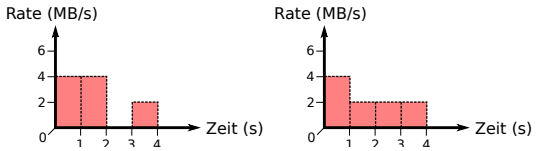

Abbildung 2.10: Ein- und Ausgangsprofil beim *Token-Bucket-Algorithmus*

von 4 MB/s für zwei Sekunden wird durch den *Token-Bucket* auf eine Sekunde mit 4 MB/s beschränkt. Die übrigen Daten des Bursts und die in der Folge eingehenden Daten werden mit der kontinuierlichen Rate von 2 MB/s gesendet.

Zusammenfassend bietet der *Token-Bucket-Algorithmus* die Möglichkeit zur Begrenzung von Bursts mit einer anschließenden kontinuierlichen maximalen Senderate. Wird diese Rate nicht komplett ausgenutzt, wird Kapazität für einen erneuten Burst angespart. In der Folge wird der *Token-Bucket* eine zentrale Rolle zur Umsetzung von *QoS* einnehmen.

2.4 Quality of Service

2.4.1 Anforderungen von Anwendungen ans Netz

Das grundlegende Ziel von *QoS* ist es Datenströmen eine gewisse Qualität zu garantieren. Das heißt, dass im Überlastfall Datenströme, denen eine Qualitätsgarantie ausgesprochen wurde, diese auch eingehalten werden. Dementsprechend werden die Datenströme, die einen *Best Effort Service* nutzen, also nur bestmöglich behandelt werden, noch stärker limitiert. Der Begriff Qualität bezieht sich dabei auf die Merkmale Zuverlässigkeit, Latenz, Jitter und Bandbreite einer Verbindung.
Unter Zuverlässigkeit wird in diesem Zusammenhang die Korrektheit der Daten nach der Übertragung verstanden. Im Videostreaming ist es beispielsweise nicht von großer Bedeutung, wenn ein Paket korrumpiert wurde und dadurch ein Bildfehler auftritt. Bei Authentifizierungen wird hingegen ein hohes Maß an Zuverlässigkeit benötigt, da sie bei jedem Fehler fehlschlagen.
Durch eine erhöhte Latenz wird die Verwendung einer Echtzeitanwendung, wie die Internettelefonie, stark eingeschränkt. Bei Dateiübertragungen oder Streams wird die Funktion trotz hoher Latenz hingegen kaum eingeschränkt, da kurze Wartezeiten zu Beginn kein Problem darstellen.
Durch einen erhöhten Jitter wird bei Streams hingegen eine enorme Störungen verursacht. Als Jitter werden die Schwankungen der Latenzen der einzelnen Pakete eines Datenstroms bezeichnet. Oft wird der Jitter als Standardabweichung der Latenzen angegeben. Für einen Audiostream wird durch einen hohen Jitter eine unregelmäßige Unterbrechung der Wiedergabe ausgelöst. Dies geschieht, da durch das verzögerte Eintreffen der Pakete, die zur Wiedergabe benötigten Daten nicht rechtzeitig vorliegen.
Schließlich wird von den Anwendungen unterschiedlich viel Bandbreite benötigt. Müssen viele Daten, wie etwa beim Videostreaming, versendet werden, so ist auch die benötigte Bandbreite hoch. Bei Anwendungen mit wenig Datenaufkommen wird entsprechend wenig benötigt. Eine Übersicht zu den Anforderungen unterschiedlicher Dienste ist in der Tabelle in Abbildung 2.11 dargestellt.

Anwendungen werden je nach Anforderung an die vier Merkmale in unterschiedliche Klassen eingeteilt. Der Klasse *konstante Bitübertragungsrate* werden Anwendungen, wie etwa die Internettelefonie, zugeordnet. Werden in einem Datenstrom viele unterschiedlich große Pakete mit einer hohen Anforderung an Latenz und Jitter übertragen, wird die entsprechende Anwendung in die Klasse *variable Echtzeit-Bitübertragungsrate* eingeordnet. Nicht-Echtzeitanwendungen, die hohe Anforderungen an den Jitter haben, werden in die Klasse *variable Nichtechtzeit-Bitübertragungsrate* eingeteilt. Die letzte Klasse *verfügbare Bitübertragungsrate* wird für Anwendungen verwendet, bei denen Übertragungsverzögerung und Jitter die Funktionalität nicht sonderlich beeinflussen [20, S. 467].

Anwendung	Zuverlässigkeit	Latenz	Jitter	Bandbreite
Audiostream	gering	gering	hoch	gering
Videostream	gering	gering	hoch	hoch
VoIP	gering	hoch	hoch	gering
Dateiübertragung	hoch	gering	gering	hoch
Authentifizierung	hoch	mittel	mittel	gering
Webbrowser	mittel	mittel	gering	mittel

Abbildung 2.11: Anforderungen von Anwendungen an das Netz [20, S. 466, Abb. 5.27]

Wird unabhängig von Anwendungen das gesamte Netz betrachtet, zeigt sich, dass vier zentrale Punkte diskutiert werden müssen, damit *QoS* adäquat genutzt werden kann.
Der erste Punkt, die *Anforderungen der Anwendung an das Netz*, wurde im Laufe dieses Abschnitts beschrieben und anhand der Tabelle in Abbildung 2.11 für einige Beispiele zusammengefasst.
Zur *Regulierung des Verkehrs bei Netzeintritt* wurden in Kapitel 2.3 zwei Algorithmen vorgestellt, die eine nachhaltige Beeinflussung des Zuflusses ermöglichen. Im folgenden Abschnitt 2.4.2 wird auf dieser Basis eine Abstraktion zur formellen Beschreibung eines Netzwerksystems vorgenommen.
Durch die anderen zwei Punkte werden die Fragen nach der *Reservierung von Ressourcen auf einem Router zur Garantie von Performance* und die der *Aufnahme von zusätzlichem Verkehr* aufgeworfen. Dafür werden in Abschnitt 2.4.3 drei Hauptprinzipien zur Realisierung von *QoS* vorgestellt. Auf dieser Basis werden in den Abschnitten 2.4.4 und 2.4.5 zwei Architekturen vorgestellt, die unterschiedliche Herangehensweisen an die Umsetzung von *QoS* bieten.

2.4.2 Service Modell

Damit durch *QoS* echte Garantien ausgesprochen werden können, ist eine formale Beschreibung und eine mathematische Betrachtung des Ablaufs unumgänglich. Die hier dargestellten Ergebnisse basieren auf dem Werk *Network Calculus* von *Le Boudec* [15].

Für die formale Beschreibung wird zunächst ein einzelnes System, beispielsweise ein Router, betrachtet. In Abbildung 2.12 sind zwei Funktionen eingezeichnet. Die *blaue* Funktion $R(t)$ spiegelt den Eingangs- und die *rote* Funktion $R^*(t)$ den

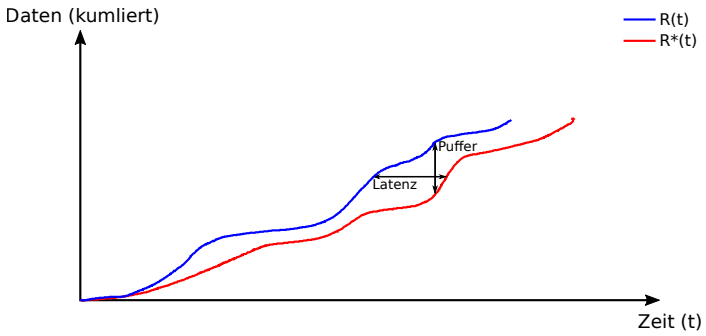

Abbildung 2.12: Ein- und Ausgangsprozess an einem Router

Ausgangsprozess mit $t \geq 0$ wieder. Als Eingangsprozess werden die an einem System eingehenden Daten in Abhängigkeit zur vergangenen Zeit bezeichnet, entsprechend die von einem System verarbeiteten Daten in Abhängigkeit zur Zeit als Ausgangsprozess.
Im Beispiel werden beide Prozesse als stetige Funktionen der kumulierten Daten gegenüber der Zeit interpretiert. Demnach wird ein kontinuierliches Modell angewendet. Das heißt, dass davon ausgegangen wird, dass die Daten stetig eingehen und nicht separate Pakete betrachtet werden. Mit Eingang eines Datums kann direkt mit der Bearbeitung begonnen werden.
Aufgrund der beiden Prozesse kann für jedes Datum die Menge der sich im System befindlichen Daten und die Latenz berechnet werden. Formal ausgedrückt wird die Menge x der Daten zum Zeitpunkt t im System durch

$$x(t) = R(t) - R^*(t)$$

beschrieben. Im kontinuierlichen Modell kann dies grafisch, wie in Abbildung 2.12 eingezeichnet, als vertikaler Abstand der Funktionen $R(t)$ und $R^*(t)$ dargestellt werden. Unter der Latenz wird in diesem Zusammenhang die Zeit verstanden, die vergeht, bis ein empfangenes Bit bearbeitet wird, wenn alle zuvor eingegangenen Pakete auch zuvor bearbeitet werden. Formal wird die Latenz l zum Zeitpunkt t durch

$$l(t) = \inf\{\tau \geq 0 : R(t) \leq R^*(t + \tau)\}$$

bestimmt. Werden, wie im Beispiel, nur steige Funktionen betrachtet, entspricht dies genau dem in der Abbildung eingezeichneten horizontalen Abstand.
Ziel des *Service Modells* ist es durch eine möglichst gute Abschätzung eine Beziehung zwischen dem Eingangs- und Ausgangsprozess zu finden. Das heißt, dass eine maximale Durchlaufzeit eines Paketes durch ein System, wie zum Beispiel einem Router, gefunden werden soll.

Für den Eingangsprozess muss eine Abschätzung nach oben gefunden werden. Eine mögliche Schranke kann über zwei grundlegende Faktoren, der maximalen *Burst-Size* und der maximalen *Senderate*, definiert werden. Das sind genau

die zwei Faktoren, die durch das in Abschnitt 2.3.2 beschriebene *Token-Bucket-Prinzip* kontrolliert werden können. Im kontinuierlichen Modell entspricht ein *Token-Bucket* einer Funktion der Art:

$$\gamma_{\xi,\sigma}(t) = \begin{cases} \xi t + \sigma & \text{für } t > 0 \\ 0 & \text{für } t = 0 \end{cases}$$

Dabei entspricht ξ der Senderate und σ der maximalen *Burst-Size*.In Abbildung 2.13 sind zwei Abschätzungen an den Eingangsprozess $R(t)$ angelegt. Die *Burst-*

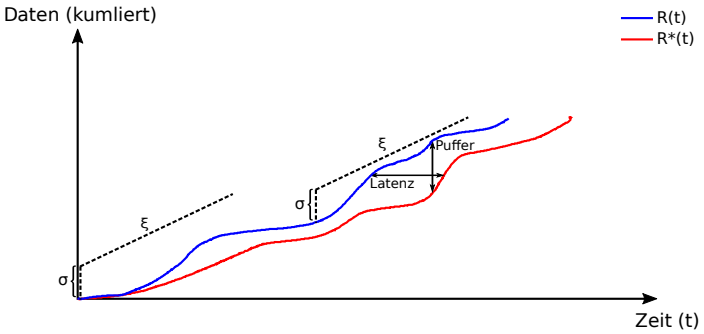

Abbildung 2.13: Obere Schranke des Eingangsprozesses

Size σ ist vertikal abgetragen, da das Senden auf einen Schlag erfolgen kann. Daran angeschlossen ist die konstante Senderate ξ. Es wird deutlich, dass der Eingangsprozess durch die beiden Faktoren nach oben beschränkt wird. Gilt die Abschätzung unabhängig von der Stelle an der sie angelegt wird immer, wird sie auch *Arrival Curve* genannt. Für das kontinuierliche Modell gilt demnach formal, dass eine Funktion α für $t \geq 0$ eine *Arrival Curve* von einem Eingangsprozess $R(t)$ ist, wenn für alle t gilt:

$$\forall s \leq t : R(t) - R(s) \leq \alpha(t - s)$$

Die Definition der *Arrival Curve* über den *Token-Bucket* hat einen weiteren Vorteil: Funktionen der Art $\gamma_{\xi,\sigma}(t)$ sind subadditiv und es gilt $\gamma_{\xi,\sigma}(0) = 0$. *Le Boudec* definiert diese Funktionen als *gute Funktionen* [15, S. 14, Def. 1.2.4]. Wird ein Eingangsprozess durch zwei *gute Funktionen* begrenzt, sind also beide *Arrival Curves*, kann über ihren subadditiven Abschluss eine bessere *Arival Curve* gefunden werden. Die Berechnung des subadditiven Abschlusses kann mittels der *min-plus Faltung* (im Folgenden durch $\otimes$ dargestellt) durchgeführt werden. Im kontinuierlichen Modell entspricht die *min-plus Faltung* von Funktionen der Bildung des Infimums. Für zwei positive und nicht fallende Funktionen f und g, wie sie auch nach der Definition von $R(t)$ und $R^*(t)$ vorliegen, gilt daher:

$$f(t) \otimes g(t) = \inf_{0 \leq s \leq t} \{f(t - u) + g(u)\}$$

Zur Abschätzung des Ausgangsprozesses wird die maximale Durchlaufzeit eines Paketes durch das System gesucht. Dies entspricht der Latenz, die durch die Komponente verursacht wird. Bei Betrachtung eines Routers ergibt sich die maximale Latenz aus der maximalen Verzögerung innerhalb der Warteschlange, sowie der minimalen Bedienrate, mit der die Warteschlange abgearbeitet wird. Eine darüber definierte Abschätzung wird *Rate-Latency Curve* genannt. Im kontinuierliches Modell ergibt sich zur *Rate-Latency Curve* eine Funktion der Art:

$$\chi_{\rho,\kappa}(t) = \rho(t - \kappa)$$

Dabei entspricht ρ der minimalen Bedienrate und κ der maximalen Verzögerung der Warteschlange. In Abbildung 2.14 sind zwei *Rate-Latency Curves* eingezeichnet. Die Verzögerung κ wird horizontal an den Eingangsprozess $R(t)$ angelegt, da

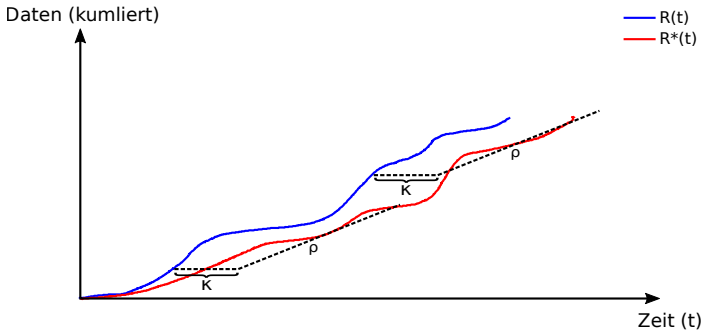

Abbildung 2.14: Untere Schranke des Ausgangsprozesses

im schlechtesten Fall ein Paket auf eine komplett gefüllte Warteschlange stoßen kann. Daran angeschlossen wird die minimale Bedienrate ρ des Routers, mit der die Pakete bearbeitet werden.
Wird der gesamte Eingangsprozess durch eine Abschätzung beschrieben, so wird diese *Service Curve* genannt. Formal gilt, dass ein Eingangsprozess $R(t)$ mit einer *Service Curve* β bedient wird, wenn gilt:

$$\forall t \; \exists s : R^*(t) - R(s) \geq \beta(t - s)$$

Die *Rate-Latency Service Curve* $\beta_{\rho,\kappa}$ mit der Senderate ρ und der maximalen Verzögerung κ wird mittels einer *min-plus Faltung* aus der Abschätzung $\chi_{\rho,\kappa}(t)$ und dem Eingangsprozess $R(t)$ berechnet. Für die *Rate-Latency Service Curve* β bedeutet dies:

$$\beta_{\rho,\kappa}(t) = \inf_{0 \leq s \leq t} \{R(s) + \chi_{\rho,\kappa}(t - s)\}$$

Dadurch wird sichergestellt, dass der Abstand zwischen $R(t)$ und $\beta_{\rho,\kappa}(t)$ nie größer werden kann, als der Abstand zwischen $R(t)$ und den einzelnen Abschätzungen $\chi_{\rho,\kappa}(t)$. Somit werden die Abschätzungen auf eine *Service Curve* reduziert.

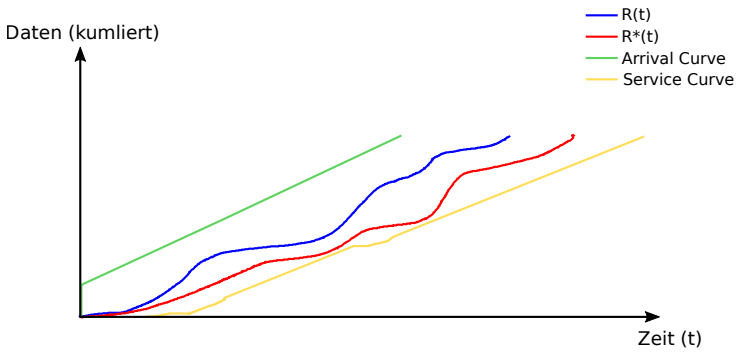

Abbildung 2.15: *Worst-Case-Abschätzung* eines Systems

Die auf das Beispiel angewendete *Rate-Latency Service Curve* ist in Abbildung 2.15 als *gelbe* und die *Arrival Curve* im Ursprung als *grüne* Funktion eingezeichnet. Dies bedeutet, dass Daten die bei einem Router mit einem Verkehrsprofil unterhalb der *grünen* Kurve eingehen, im schlechtesten Fall nach der *gelben* Kurve weitergeleitet werden. Es wurde also eine *Worst-Case-Abschätzung* der Reaktion eines Systems auf ankommende Daten gefunden und mathematisch hergeleitet. Dadurch können die Latenz und der benötigte Puffer aus der Grafik abgelesen und auch berechnet werden.

In der Regel werden die Pakete eines Datenstroms nicht nur von einer Komponente sondern über komplexe Pfade weitergeleitet. Durch das hier eingeführte Modell wird für jede weitere Komponente die *Service Curve* der vorherigen als *Arrival Curve* verwendet. Dadurch, dass für jede Komponente eine eigene Abschätzung auf Basis des Ergebnisses der vorherigen Komponente getroffen wird, summieren sich die Abweichungen für das gesamte, zusammengesetzte System zu einem ungenauen Ergebnis auf.
Die Abschätzungen können durch das *Pay Burst Only Once Prinzips* deutlich verbessert werden. Dabei wird die *Service Curve* eines zusammengesetzten Systems über die *min-plus Faltung* der für die Einzelsysteme geltenden *Service Curves* berechnet. Für das vorgestellte Modell ergibt sich dabei für zwei Systeme:

$$\beta_{\rho_1,\kappa_1} \otimes \beta_{\rho_2,\kappa_2} = \beta_{\min(\rho_1,\rho_2),\kappa_1+\kappa_2}$$

Für das Gesamtsystem ergibt sich demnach eine *Service Curve*, für die das Minimum der Raten ρ_i als Gesamtrate und die Summe der Verzögerungen durch die Warteschlangen κ_i als Gesamtverzögerung verwendet wird. Durch die bessere *Service Curve* kann direkt eine bessere Latenzschranke für das Gesamtsystem hergeleitet werden [15, Kapitel 1.4.3, S. 28f].
Als Beispiel werden im Folgenden zwei Komponenten betrachtet. Zunächst werden ihre Latenzschranken separat analysiert und nachfolgend mit der über das *Pay Burst Only Once Prinzips* ermittelten Latenzschranke verglichen. Der Eingangsstrom wird über einen *Token-Bucket* der Form $\gamma_{\xi,\sigma}(t)$ beschränkt. Die *Ser-*

vice Curves werden mit $\beta_{\rho_i,\kappa_i(t)}$ beschrieben. Zusätzlich wird zur Vereinfachung angenommen, dass $\xi \leq \rho_1$ und $\xi \leq \rho_2$ gilt.
Für das erste Einzelsystem gilt unter diesen Voraussetzungen eine Latenzschranke von:

$$L_1 = \frac{\sigma}{\rho_1} + \kappa_1 \tag{2.1}$$

Dies entspricht dem Supremum der durch β_{ρ_1,κ_1} im Vergleich zu $\gamma_{\xi,\sigma}$ entstehenden Verzögerung in jedem Punkt t. Anschaulich ist dies der größte vertikale Abstand zwischen den Funktionen $\gamma_{\xi,\sigma}$ und β_{ρ_1,κ_1}. Formal gilt demnach [15, S.23, Theroem 1.4.2]:

$$\begin{aligned} L &= \sup\{\delta(s)\},\ mit \\ \delta(s) &= \inf\{\tau \geq 0 : \gamma(s) \leq \beta(s+\tau)\} \end{aligned}$$

Daraus resultiert eine *Arrival Curve* für das zweite System von:

$$\alpha(t) = \sigma + \xi * (t + \kappa_1) \tag{2.2}$$

Wird dafür wiederum eine Latenzschranke berechnet, so ergibt sich:

$$L_2 = \frac{\sigma + \xi * \kappa_1}{\rho_2} + \kappa_2 \tag{2.3}$$

Dabei fällt im Vergleich zwischen den Ergebnissen für die Latenzschranken 2.1 und 2.3 der hinzugekommene Faktor $\frac{\xi*\kappa_1}{\rho_2}$ auf. Dieser entsteht, da durch das erste System der *Burst* verstärkt worden ist. Formal wird der Faktor über die *Arrival Curve* 2.2 an der Stelle $t = 0$ erzeugt. Für das Gesamtsystem ergibt sich eine Latenzschranke aus den Ergebnissen 2.1 und 2.3 von:

$$L = L_1 + L_2 = \frac{\sigma}{\rho_1} + \frac{\sigma + \xi * \kappa_1}{\rho_2} + \kappa_1 + \kappa_2 \tag{2.4}$$

Werden die beiden Komponenten über das *Pay Burst Only Once Prinzip* zu einem System zusammengefasst, wird zunächst die gemeinsame *Service Curve* über die *min-plus Faltung* berechnet. Daraus kann direkt eine Latenzschranke für das Gesamtsystem berechnet werden:

$$L = \frac{\sigma}{\min\{\rho_1, \rho_2\}} + \kappa_1 + \kappa_2 \tag{2.5}$$

Im Vergleich zwischen den beiden Ergebnissen für L (2.4 und 2.5), stellt sich heraus, dass in der ersten Berechnung zwei Einträge des Typs $\frac{\sigma}{\rho_i}$ summiert werden. In der zweiten Berechnung taucht dieser Eintrag nur einmal auf. Der Eintrag beschreibt die Latenz, die durch einen möglichen Burst aus dem *Token-Bucket* entsteht. Bei einer Verkettung der beiden Einzelsysteme fällt dieser Betrag nur

einmal an. Diese Vorgehensweise lässt sich auf beliebig viele Systeme ausweiten. Darin liegt auch der Ursprung der Namensgebung des *Pay Burst Only Once Prinzips*.

Durch die genauen Abschätzungen wird ein Teil der Basis für die Umsetzung von *QoS* geschaffen. Es wurde gezeigt, dass anhand eines festen maximalen Verkehrsprofils und der Reaktionszeit der beteiligten Systeme, die Latenz und Bandbreite zwischen Systemen begrenzt werden kann. Im folgendem Abschnitt 2.4.3 werden drei weitere Hauptprinzipien zur Umsetzung von *QoS* betrachtet.

2.4.3 Prinzipien zur Umsetzung von QoS

Durch die drei Prinzipien, *Paketklassifikation*, *Isolation von Scheduling und Policing* und *Zugangssteuerung* wird *QoS* auf Basis von Datenströmen realisiert.

Die *Paketklassifikation* wird zur leichteren Unterscheidbarkeit und Zuordnung der einzelnen Pakete zu den entsprechenden Datenströmen verwendet. Aufgrund dessen kann eine Entscheidung über den Grad der Priorisierung getroffen werden. Bei *IP-Paketen* wird zur Klassifizierung das Header-Feld *Type of Service (ToS)* verwendet. Je nach Bytecode werden andere Strategien angewendet.

Aufgrund der Klassifizierung kann die *Isolation von Scheduling und Policing* umgesetzt werden. Das *Policing* wird eingesetzt, um sicherzustellen, dass die Senderaten der einzelnen Hosts eingehalten werden. Dabei werden die *Durchschnittsrate*, die *Spitzenrate* und die *Burst Size* kontrolliert. Dafür werden Maßnahmen des *Traffic Shaping* (vgl. Kapitel 2.3) verwendet. Durch das *Scheduling* werden die Reaktion der Router auf eingehende Pakete definiert. In den meisten Fällen wird entweder ein *Priority-Scheduling* oder ein *Weighted Fair Queuing (WFQ)*, verwendet.
Im *Priority-Scheduling* werden zwei getrennte Warteschlangen, eine für Pakete mit hoher Priorität und eine für alle anderen, definiert. Solange Pakete in der Warteschlange mit hoher Priorität vorhanden sind, werden diese gesendet. Nur wenn diese Warteschlange leer ist, werden andere Pakete verschickt.
Als Basis für *WFQ* wird ein *Round Robin Verfahren* eingesetzt. Es werden beliebig viele Klassen, denen jeweils eine Übertragungsrate garantiert werden kann, unterstützt. Jeder Klasse wird eine eigene Warteschlange, sowie ein Gewicht ω_i zugeordnet. In jedem Sendezyklus werden alle Klassen nacheinander durchlaufen und entsprechend ihres Gewichtes ein Anteil der Senderate eingeräumt. Sollte eine Warteschlange leer sein, wird sie übersprungen. Jeder Klasse wird dadurch ein minimaler Anteil der Rate von $\omega_i / \sum_{q=0}^{n} \omega_q$ garantiert.
Neben dem *Scheduling* muss auch die *Discard Policy* betrachtet werden. Hierbei existieren wiederum drei unterschiedliche Ansätze, die jeweils andere Einflüsse auf die Datenströme haben. Beim *Tail Drop* werden alle Pakete gelöscht, die

empfangen werden, während die entsprechende Warteschlange voll ist. Durch den *Priority Drop* werden Pakete anhand einer Prioritätenliste verworfen und durch *Random Drop* werden schon bevor eine Warteschlange voll ist, zufällig Pakete nicht angenommen.

Durch die *Zugangssteuerung* wird geregelt wann und ob eine Verbindung aufgenommen werden kann. Dabei werden unterschiedliche Bedingungen überprüft. Diese umfassen beispielsweise eine grundlegende Berechtigung zur Nutzung von Prioritäten. Eine weitere Aufgabe ist die Reservierung der angeforderten Ressourcen. Mittels dieser Reservierung wird sichergestellt, dass die erforderlichen Garantien auch im Netz eingehalten werden können. Ansonsten wird die neue Verbindung nicht akzeptiert.

Im Laufe der Entwicklung von *QoS* haben sich die *Integrated Service Architektur* und die *Differentiated Service Architektur* durchgesetzt. Sie bieten unterschiedliche Herangehensweisen an die hier erläuterten Prinzipien. Sie werden gesondert in den Abschnitten 2.4.4 und 2.4.5 betrachtet.

Grundsätzlich wird die Nutzung von *QoS* nur in Überlastsituation sichtbar. Nur dann wird die Einschränkung durch die Prioritätsklassen spürbar, da ansonsten keine Limitierung der Datenströme notwendig ist. Natürlich entsteht durch *QoS* keine zusätzliche Bandbreite, es geht bei entsprechender Wahl der Algorithmen aber auch keine verloren. Daraus folgt, dass die niedrig priorisierten Klassen eingeschränkt werden, um den Klassen mit hoher Priorität Bandbreite für ihre Übertragung zu garantieren.

2.4.4 Integrated Service Architecture

Die *Integrated Service Architecture (IntServ)* wurde von der *Internet Engineering Task Force (IETF)* zur Bereitstellung von Dienstgarantien in Netzen auf IP-Basis im Jahr 1994 standardisiert. Ziel der Architektur war die Integration von Multimediastreams ins Internet. Das zu Grunde liegende Konzept ist eine Ende-zu-Ende Reservierung von Ressourcen für jeden einzelnen Datenstrom in allen beteiligten Routern. Die Art und Weise der Implementierung zur Umsetzung der Architektur wurde nicht weiter spezifiziert.

Für die unterschiedlichen Anforderungen der Verbindungen werden durch *IntServ* drei Servicemodelle definiert, die unter dem Begriff *Request Specification (R-Spec)* zusammengefasst werden. Harte Garantien der Verbindung gegenüber werden mit dem Modell *Guaranteed Service* zur Verfügung gestellt. Das heißt, dass eine minimale Bandbreite, sowie eine maximale Latenz bei Bedarf genutzt werden können. Wird eine garantierte Latenz nicht zwingend benötigt, wird das Modell

Controlled Load verwendet. Dadurch wird Bandbreite garantiert, Jitter aber zugelassen. Schließlich werden alle Pakete von Verbindungen, die das Modell *Best Effort* verwenden, zum Auffüllen ungenutzter Kapazität verwendet.
Alle weiteren Parameter zur Beschreibung eines Datenflusses werden in der *Traffic Specification (T-Spec)* zusammengefasst [19]. Üblicherweise handelt es sich dabei um die Parameter, die die Arbeitsweise eines *Token Buckets* beschreiben. Die genaue Funktionsweise wurde bereits in Abschnitt 2.3.2 erläutert.

Bevor eine Garantie für eine Verbindung ausgesprochen werden kann, muss eine Zugangskontrolle stattfinden. Dabei wird die technische Realisierbarkeit der aus *R-Spec* und *T-Spec* bestehenden *Flow Specification (FlowSpec)* kontrolliert. Parallel dazu wird durch die *Policy Control*, anhand der sogenannten *Filter Specification (FilterSpec)* überprüft, ob der Initiator die entsprechende Berechtigung besitzt eine solche Reservierung vorzunehmen. Unter die *FilterSpec* fallen vor allem die Quelladresse und der Quellport, andere Parameter zur Überprüfung sind zusätzlich möglich [8].

Für die Realisierung der Ende-zu-Ende Reservierung wird ein zusätzliches Signalisierungs- und Kontrollprotokoll benötigt. Durch das Protokoll werden die erforderlichen Parameter auf dem Routingpfad vom Sender zum Empfänger allen Routern mitgeteilt. Die tatsächliche Reservierung auf den beteiligten Systemen erfolgt aufgrund dieser Nachrichten, ist aber nicht Teil des Protokolls [21].
Bei *IntServ* wird das *Resource Reservation Protocol (RSVP)* verwendet. Darin wird in der Regel eine Reservierung durch den Sender ausgelöst. Dazu wird eine sogenannte *Path-Nachricht* verschickt. Darin werden Informationen über die spätere Datenverbindung, Pfadinformationen und der Empfänger übertragen. Explizit enthält die Nachricht einerseits alle *T-* und *R-Spec* Informationen, andererseits je ein Feld für einen *Hop-Count* und eine *kumulierte Latenz*.
Wird eine *Path-Nachricht* von einem Router empfangen, wird der *Hop-Count* um eins inkrementiert. Zusätzlich wird durch den Router eine Latenzgarantie, die für die Verbindung ausgesprochen werden kann, bestimmt. Diese wird auf den kumulierten Wert aufaddiert. Mit den neuen Werten wird die Nachricht anhand der Adresse des Empfängers weitergeleitet. Dadurch wird gewährleistet, dass die *Path-Nachricht* den gleichen Weg durchs Netz nimmt, wie auch die späteren Pakete. Sollte eine Reservierung mit den gegebenen Parametern nicht möglich sein, wird eine negative *Resv-Nachricht* an den Sender geschickt und somit die Verbindung abgelehnt.
Die auf dem Weg folgenden Router arbeiten analog. Erhält schließlich der Empfänger eine *Path-Nachricht* wird eine Entscheidung über die tatsächlich benötigten Werte der Parameter getroffen. Diese werden mittels einer *Resv-Nachricht* auf dem gleichen Weg zum Sender geschickt. Aufgrund der *R-Spec* Informationen in der Nachricht wird von jedem Router entlang des Routingpfades eine Reservierung für die Verbindung angelegt. Die Nachricht wird entsprechend angepasst

und weitergeleitet. Erreicht die Nachricht den Sender, wird anhand der *T-Spec* Informationen ein *Token Bucket* definiert. Daraufhin kann mit der Übertragung der Daten begonnen werden.
Dem Protokoll liegt ein *Soft-State-Modell* zu Grunde. Daher müssen zur Aufrechterhaltung der Reservierung in periodischen Abständen weitere *Path-Nachrichten* versendet werden [9].

Eine mögliche Realisierung von *IntServ* auf einem Router ist in der Abbildung 2.16 dargestellt. Darin wird die zur Verfügung stehende Bandbreite mittels *WFQ*

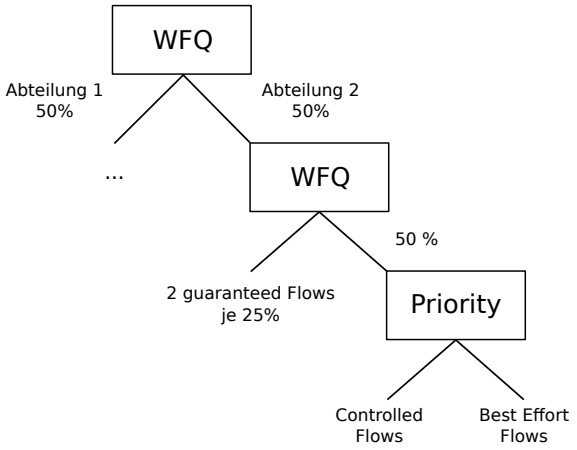

Abbildung 2.16: Aufteilung der Bandbreite mittels *IntServ*

zwischen zwei Abteilungen in gleichen Teilen aufgeteilt. In der zweiten Abteilung werden zwei Anwendungen als *Guaranteed Flows* priorisiert. Ihnen werden jeweils 25 % der Bandbreite der zweiten Abteilung, sowie eine harte Latenzschranke über ein erneutes *WFQ* eingeräumt. Die restlichen 50 % der Bandbreite der zweiten Abteilung werden mittels *Priority Scheduling* an *Controlled* und *Best Effort Flows* verteilt. Daher werden nur Daten aus den *Best Effort Flows* weitergeleitet, wenn keine Daten von *Controlled Flows* auf Bearbeitung warten.

Zusammenfassend wird durch *IntServ* die Vergabe von harten Dienstgarantien ermöglicht. Durch die strikte Trennung aller Datenströme in jeder Komponente des Netzes kann zudem das vorgestellte *Pay Burst Only Once Prinzip* angewendet werden. Dadurch ist eine deutlich bessere Abschätzung der jeweils benötigten Kapazität eines Datenstroms und somit auch eine effektivere Aufteilung von Bandbreite und Latenz möglich.
Allerdings bietet die Architektur neben dem *Guaranteed Service* nur eine weitere Unterscheidungsmöglichkeit. Weitere Abstufungen zwischen *Controlled Load* und *Best Effort* wären wünschenswert. Der größte Nachteil von *IntServ* ist die, für jede Verbindung zwingend notwendige, dynamische Reservierung von Ressourcen auf jedem Router zwischen den Hosts. Einerseits können dadurch Skalierungsprobleme in Bereichen mit vielen Verbindungen entstehen. Andererseits müssen

auch Anpassungen auf Routern vorgenommen werden, die nicht im eigenen Besitz sind. Daher muss einer generellen Berechtigung dazu im Vorhinein von anderen Parteien zugestimmt werden.
Erfolgt eine Umsetzung von *QoS* im eigenen, privaten Netz oder in einem Intranet, so bietet *IntServ* hierfür eine ausreichende und im Bezug auf harte Dienstgarantien eine sehr mächtige Funktionalität.

2.4.5 Differentiated Service Architecture

Die *Differentiated Services Architecture (DiffServ)* bietet die Möglichkeit viele unterschiedliche Prioritätsgruppen zu definieren. Die Basis dafür wurde im Jahr 1998 von der *IETF* geschaffen. Je nach Klassifizierung der Pakete wird auf eine andere Behandlungsstrategie zurückgegriffen. Im Gegensatz zu *IntServ* werden die Konfigurationen statisch vorgenommen, sodass die aufwändige Signalisierung entfällt [6].

Die Markierung der Pakete wird über eine Neudefinition des Feldes *Type of Service* im IP-Header umgesetzt. Das sogenannte *Differentiated Service Feld* ist in Abbildung 2.17 schematisch dargestellt.

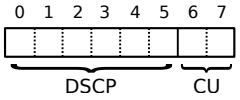

Abbildung 2.17: Das *Differentiated Service Feld*

Die ersten sechs Bits bilden den *Differentiated Services Codepoint (DSCP)*. Die letzten beiden werden für *DiffServ* nicht verwendet. Über den *DSCP* werden die Pakete ihrer jeweiligen Klasse zugeordnet.
Demnach ist es möglich 64 verschiedene Prioritätsklassen zu konfigurieren. Die Behandlungsregeln, die zur eigentlichen Umsetzung im Router angewendet werden, werden auch *Per Hop Behavior (PHB)* genannt. In der Spezifikation von *DiffServ* werden drei Klassen mit einem entsprechenden *DSCP* empfohlen, einzig ein *Default PHB* ist vorgeschrieben. Der Default entspricht meistens einem *Best Effort Service*.
Eine Empfehlung ist das *Class Selector PHB (CS PHB)*. Es dient zur Wahrung der Rückwärtskompabilität zur ursprünglichen Verwendung von *ToS* bei *IPv4* [16].
Zur weiteren differenzierten Priorisierung werden die Klassen *Expedited Forwarding PHB (EF PHB)* und *Assured Forwarding PHB (AF PHB)* empfohlen. Dem *EF PHB* wird eine geringe Latenz, hohe Zuverlässigkeit und ein geringer Jitter zugeordnet, solange die Senderate in einem definierten Rahmen bleibt. Es wird in der Regel als Premiumdienst angeboten. Zur erfolgreichen Garantie der Eigenschaften muss sichergestellt werden, dass die Bedienrate für die entsprechend

markierten Pakete größer ist, als deren Ankunftsrate. Dadurch wird verhindert, dass Stauungen in den Warteschlangen entstehen. Des Weiteren muss garantiert werden, dass Pakete nicht zu lange in einer Warteschlange aufgehalten werden. Daraus resultiert ein geringes Delay und ein geringer Jitter. Zu Verhinderung von Paketverlusten muss gewährleistet werden, dass die Warteschlangen nur leicht gefüllt oder leer sind. Zur Umsetzung muss eine strenge Zugangskontrolle realisiert werden. Gegebenenfalls müssen Verbindungen abgelehnt werden, um die Warteschlangen in den Routern kurz zu halten. Außerdem muss eine schnelle Reaktion der Router auf eingehende Pakete erfolgen. Dazu wird in der Regel ein *Priority Scheduling* verwendet. Daher muss zusätzlich darauf geachtet werden, dass durch den Verkehr des *EF PHB* nicht das gesamte Netz ausgelastet wird, da sonst keine oder nur wenige andere Pakete übertragen werden könnten[11].
Eine feinere Einteilung wird von dem *AF PHB* geboten. Es werden unterschiedliche Klassen mit jeweils verschiedenen *Discard Policies* vorgegeben. Die genauen Anzahlen werden im Standard nicht festgelegt. Für die generelle Verwendung werden vier Klassen mit jeweils drei verschiedenen *Discard Policies* empfohlen. Je nach Klasse und Strategie werden die Pakete unterschiedlich markiert. Die Klassen bieten untereinander eine unabhängige Behandlung der Pakete. Durch den Standard wird grundsätzlich keine Priorisierung zwischen den Klassen vorgenommen. Eine unterschiedliche Behandlung wird durch die Scheduling-Strategie umgesetzt. Die *Discard Policies* innerhalb einer Klasse haben keinen Einfluss auf die Reihenfolge in der Warteschlange und somit auch nicht auf die Sendereihenfolge. Ziel der unterschiedlichen *Discard Policies* ist es, bei einer zu langen Warteschlange eine weitere Priorisierungsmöglichkeit zu schaffen. Die Auswirkung unterschiedlicher *Discard Policies* in einer Klasse ist in Abbildung 2.18 beispielhaft an zwei Strategien dargestellt. Darin wird die Wahrscheinlichkeit, mit der ein Paket einer Strategie verworfen wird, gegen die Durchschnittliche Länge der Warteschlange der Klasse aufgetragen. Die Pakete, die nach der *grünen*

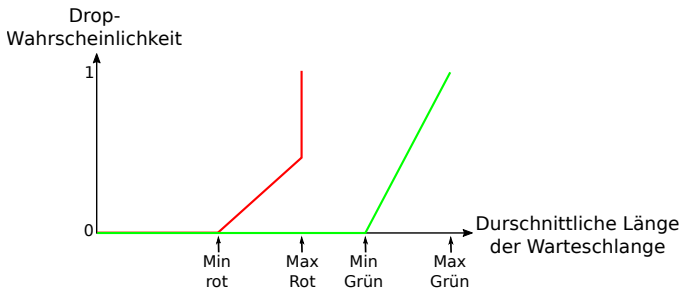

Abbildung 2.18: Auswirkung unterschiedlicher *Discard Policies*

Strategie verarbeitet werden, sind höher priorisiert. Niedrig priorisierte Pakete werden nach der *roten* Strategie behandelt. Generell wird dabei das sogenannte *Random Early Drop (RED)* verwendet. Dabei wird relativ früh damit begonnen Pakete der *roten* Strategie zu verwerfen. Dadurch wird versucht die gemeinsame

Warteschlange zu entlasten. Reicht dies nicht aus werden ab einem bestimmten Punkt alle Pakete der *roten* Strategie verworfen. Sollten immer noch mehr Pakete ankommen als weitergeleitet werden können, so werden zusätzlich auch Pakete der *grünen* Strategie verworfen. Die Wahrscheinlichkeit wird mit zunehmender Länge der Warteschlange weiter erhöht bis schließlich alle ankommenden Pakete verworfen werden [13].

Zur Vereinfachung der Architektur wird in *DiffServ* eine Unterscheidung zwischen den Routern getroffen. Die Router, die eine Verbindung zu einer anderen Domäne haben, werden *Zugangs-Router* genannt. Diesen Routern werden vor allem Datenstrom spezifische Aufgaben zugeteilt. Dabei werden die Pakete meist verbindungsspezifisch klassifiziert. Jeder Klasse wird durch ein Policing-Mechanismus ein maximales Lastprofil auferlegt. Aufgrund der Klassifizierung und des angeforderten Services kann schließlich ein entsprechendes *Shaping*, das Setzen des *DSCP-Feldes* oder auch das Verwerfen der Pakete erfolgen. Werden die Pakete durch den *Zugangs-Router* weitergeleitet, haben diese die Erlaubnis den angeforderten Service zu nutzen und entsprechen dem vorgesehenen Lastprofil.
Aufgrund dieser Vorarbeitet können alle Router in der Domäne, den sogenannten *Kernnetz-Routern*, einer statischen Konfiguration unterliegen. Es wird pro Klasse, daher pro genutztem *DSCP*, ein *PHB* konfiguriert [16].

Eine mögliche Konfiguration zur Verteilung der Bandbreite ist in Abbildung 2.19 dargestellt. Mittels eines *Priority-Schedulings* wird darin ein *EF PHB* isoliert.

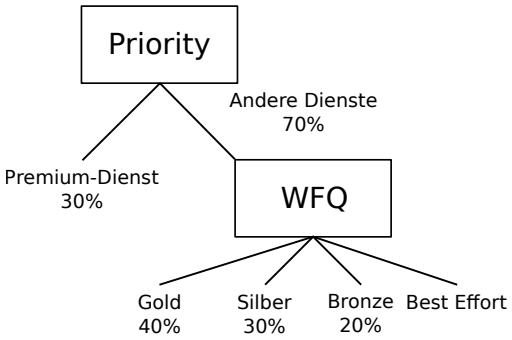

Abbildung 2.19: Aufteilung der Bandbreite mittels *DiffServ*

Der Rest des Verkehrs wird in die Prioritätsklassen Gold, Silber und Bronze, sowie eine *Best Effort* Default-Klasse aufgeteilt. Dazu wird ein *WFQ* verwendet. Auf eine weitere Unterscheidung innerhalb der Prioritätsklassen wird verzichtet. Die Zugehörigkeit eines Paketes zu einer Prioritätsklasse wird durch den *DSCP* festgelegt.

Mit der bisherigen statischen Konfiguration der *Zugangs-Router* wird bei jedem neu genutzten Dienst eine neue Analyse des Verkehrs notwendig, da durch die

Verkehrsprofile direkt die Leistungen der *Kernnetz-Router* beeinflusst werden. Damit Dienste auch dynamisch genutzt werden können, existiert eine zusätzliche Kontrolleinheit, der *Bandwidth Broker*.
Durch den *Bandwidth Broker* wird die Instanz der Zugangskontrolle gebildet und bei Bedarf die Konfiguration der *Zugangs-Router* geändert. Die Konfiguration der *Kernnetz-Router* wird davon nicht beeinflusst. Durch die zentrale Steuerung wird auch eine leichtere Authentifizierung der Nutzer anhand ihrer Identität, der Anforderungen und des aktuellen Zustands des Netzes ermöglicht.
Ein weiteres Problem, das durch den *Bandwidth Broker* gelöst wird, ist die Kommunikation zwischen Domänen. Sollen Dienstangebote zwischen einzelnen *DiffServ-Domänen* genutzt werden, ist die Basis jeweils ein *Service Level Agreement*. Dies ist ein Vertrag zwischen den Betreibern der Domänen. Zur Initialisierung wird durch ein externes Programm eine Kommunikation zwischen den verantwortlichen *Bandwidth Brokern* aufgebaut, damit die benötigten Konfigurationen umgesetzt werden können. Dadurch ist die Nutzung von *DiffServ* in Verbindung mit Ende-zu-Ende-Diensten über Domänen hinweg möglich [17].

Zusammenfassend wird durch *DiffServ* eine individuell konfigurierbare Möglichkeit Datenverkehr zu separieren und Dienstgarantien auszusprechen geboten. Für die Dienstgarantien wird eine komplexe Zugangssteuerung benötigt. Bei Einhaltung der Verkehrsprofile wird somit ein klassenbasiertes Overprovisioning erzeugt. Dadurch können Bandbreiten und Latenzen bei Einhaltung der Verkehrsprofile garantiert werden. Die Skalierungsprobleme von *IntServ* werden durch die statische Konfiguration der *Kernnetz-Router* gelöst.
Aufgrund der Zusammenfassung von Datenströmen in Klassen kann jedoch nicht auf das *Pay Burst Only Once Prinzip* genutzt werden. Dies liegt daran, dass die für die Berechnungen notwendigen *Arrival* und *Service Curves* nicht für die einzelnen Datenströme vorliegen, sondern nur für ganze Klassen. Dadurch ist keine genau Aussage über die Kapazität, die einem einzelnen Datenstrom zur Verfügung steht, möglich. Es kann lediglich eine Abschätzung der maximalen Latenz in Abhängigkeit von der zugewiesenen Bandbreite und der Anzahl an Hops vom Sender zum Empfänger getroffen werden. Diese Abschätzung wird *Charney-Bound* genannt. Sinnvolle Abschätzungen können allerdings nur bei einer geringen Anzahl von Hops (kleiner 6) und maximal 15 % der Bandbreite durchgeführt werden.

3 Quality of Service auf Infiniband

3.1 Einführung in Infiniband

Infiniband ist eine hochperformante Netzwerktechnik, die auf nachrichtenorientierten Punkt-zu-Punkt Verbindungen basiert. Infiniband wurde gegen Ende der neunziger Jahre in einem gemeinsamen Open-Source Projekt der Firmen *Intel* und *IBM* entwickelt.
Ziel bei der Entwicklung war es möglichst viel Logik auf die Netzwerkhardware auszulagern. Dadurch wird ein erheblicher Performancegewinn gegenüber anderen Techniken erzielt. Mit aktuellster Infiniband-Hardware Bandbreiten von bis zu 100 Gb/s, sowie Latenzen im Mikrosekundenbereich erreicht werden.
Für den Einsatz von Infiniband muss spezielle Netzwerkhardware verwendet werden. Das dazu benötigte Softwarepaket wird *Open Fabrics Enterprise Distribution (OFED)* genannt und über die *Open Fabrics Alliance* frei vertrieben. Zum Teil wird von den Hardwareherstellern auch eigene Software bereitgestellt. Im Folgenden wird stets auf den von *Mellanox* entwickelten *OFED-Stack* Bezug genommen. Als Grundlage für dieses Kapitel dient die von *Mellanox* veröffentlichte *Introduction to Infiniband for End Users* [12].

Der gesamte *OFED-Stack* kann, wie auch das *OSI Modell*, in unterschiedliche Schichten, die über Schnittstellen verbunden sind, eingeteilt werden. Ein Überblick über den *OFED-Stack* und ein Vergleich zur Verwendung von *IP* im herkömmlichen Sinne ist in Abbildung 3.1 dargestellt. Darin werden drei Schichten, eine Hardwareschicht und zwei Betriebssystemschichten, eingeführt. Auf der Hardware setzen herstellerspezifische Treiber auf, die auf der Seite des *OFED-Stacks* direkt aus dem Benutzerbereich des Betriebssystems angesprochen werden können. Wie im *IP-Stack* können auch bei Nutzung des *OFED-Stacks* Zwischenschichten verwendet werden.

Für den Nutzer des *OFED-Stacks* stehen demnach verschiedene Möglichkeiten zur Verfügung. In der Regel werden Schnittstellen auf Applikationsebene verwendet. Sie liefern beispielsweise eine Implementierung des *Message Passing Interfaces (MPI)* oder direkte Verbindungen zu Filesystemen. Durch die Schnittstellen wird entweder direkt auf die Hardware zugegriffen oder es werden die sogenannten *Upper Layer Protocols (ULP)* verwendet. *ULPs* sind Protokolle, die bekannte

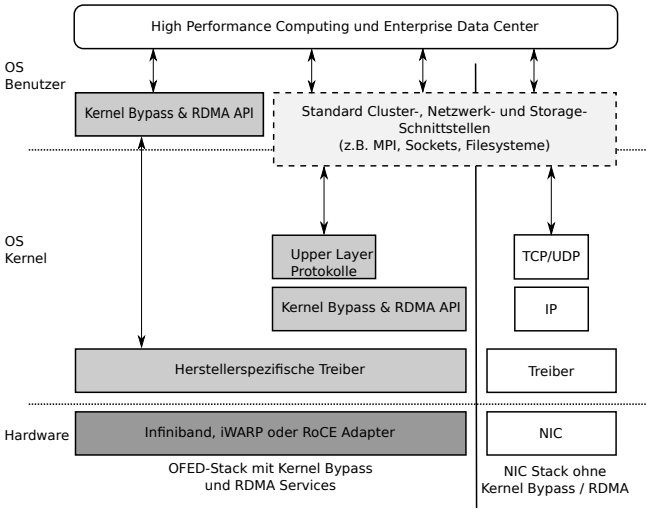

Abbildung 3.1: Schematischer Aufbau des *OFED-Stacks*

Protokolle oder häufig verwendete Funktionen auf die performante Struktur von Infiniband umsetzen. Alternativ kann auch aus einer Anwendung direkt auf *ULPs* zugegriffen werden.
Beispielsweise werden mittels *IP over Infiniband (IPoIB)* dem Nutzer die bekannten Funktion von *IP* zur Verfügung gestellt. Ein anderes Beispiel ist das *Sockets Direct Protocol (SDP)*. Es liefert die Funktionalität der *Sockets* als Kommunikationsendpunkt in Anwendungen. Generell sind durch *ULPs* viele Anwendungen ohne Anpassung auf Infiniband portierbar und können direkt genutzt werden. Wird ein *ULP* verwendet, wird zur eindeutigen Identifizierung die jeweilige *Service-ID* des *ULP* in den Header eines Infiniband-Paketes eingetragen.
Bei der Implementierung der *ULPs* oder anderer Schnittstellen werden sogenannte *Verbs* verwendet. Durch *Verbs* werden konkrete Aufgaben, wie etwa Sende- oder Empfangsoperationen, die von der Hardware ausgeführt werden sollen, repräsentiert. Die Implementierung der Verbs wird durch die Betriebssystemhersteller vorgenommen, damit sie bestmöglich ins System integriert werden können. Dabei sind die Eingangs- und Ausgangsparameter, sowie die durchgeführten Aktionen jedes Verbs in der Spezifikation von Infiniband festgelegt. Somit ähnelt die Verwendung der *Verbs* einem Funktionsaufruf mit fest definierten Zuständen.
Durch die Verwendung der *Verbs* wird die zu verrichtende Arbeit ohne Interaktion mit dem Betriebssystem direkt der Infiniband-Hardware zugewiesen. Dieser Vorgang wird *Kernel Bypass* genannt. Durch diese Umgehung des Betriebssystems wird im Vergleich zum *IP-Stack* Rechenzeit eingespart. Diese Zeit wird für die eigentliche Anwendung verwendet. Dadurch wird ein erheblicher Performancegewinn erzielt.
Sind gewünschte Funktionen nicht über *ULPs* oder andere Schnittstellen verfügbar, können *Verbs* auch direkt in Anwendungen verwendet werden. Hierfür sind

allerdings fundierte Kenntnisse der Abläufe in Infiniband zwingend notwendig.

Durch den *OFED-Stack* wird eine breite Funktionalität und eine, im Vergleich zur statischen Protokollstruktur von *IP*, große Flexibilität dem Nutzer gegenüber geschaffen. Aufgrund der verschiedenen Verwendungsmöglichkeiten kann die performante Infiniband-Struktur je nach Anforderung und Wunsch des Nutzers optimal genutzt werden.

Aufgrund der hohen Flexibilität, sowie der sehr hohen Bandbreiten und niedrigen Latenzen, kann Infiniband in vielen Bereichen eingesetzt werden. Besonders eignet sich Infiniband zum Einsatz im Bereich des *High Performance Computing (HPC)*, sowie in anderen Clustern. Aber auch in weiteren performancekritischen Umgebungen, wie zum Beispiel der Anbindung von Speicher- oder Datenbanksystemen, wird vermehrt Infiniband eingesetzt.

Im folgenden Kapitel 3.2 wird auf die grundlegenden Kommunikationstechniken, sowie auf die Überlastkontrolle, eingegangen. Darauf aufbauend wird in Kapitel 3.3 die in Infiniband umgesetzte *QoS-Architektur* vorgestellt und schließlich in Kapitel 3.4 die Konfigurationsmöglichkeiten erläutert.

Als Grundlage für die folgenden Kapitel werden das *Mellanox OFED for Linux User Manual* [5], die *Infiniband-Spezifikation* der *Infiniband Trade Association* [4], sowie das Werk *Infiniband Network Architecture* von *T. Shanley* [18] verwendet.

3.2 Kommunikations- und Kontrolltechniken

Die Endpunkte von Infiniband auf Hosts und Switches werden *Host Channel Adapter (HCA)* genannt. Die *HCAs* übernehmen die erwähnte ausgelagerte Logik, wie die Aufteilung der Daten in Pakete, sowie das Hinzufügen der benötigten Header und die üblichen von der Netzwerkhardware zu erledigenden Aufgaben.

Zur logischen Verbindung zweier Hosts wird ein *Channel* definiert. Dieser besteht auf jedem Host aus einem *Queue Pair (QP)*, das aus je einer *Send-* und einer *Receive-Queue* zusammengesetzt wird. Wird ein *Verb* aufgerufen, wird ein *Workrequest* an ein *QP* gestellt. Über diesen *Workrequest* wird die zu verrichtende Arbeit definiert und entsprechende Aktionen ausgelöst, damit die Nachrichten verschickt wird.

Werden Nachrichten in einzelne Pakete aufgeteilt, wird deren Größe durch die sogenannte *Maximum Transfer Unit (MTU)* limitiert. Die *MTU* arbeitet somit auf der Sicherungsschicht. Sie kann für jeden Port separat gesetzt werden, sollte aber homogen gehalten werden. Maximal kann ein Wert von 4096 Bytes verwendet werden. Dadurch wird der maximale *Packet-Payload* festgelegt. Das sind die Daten, die zwischen allen Transport-Headern und den Prüfsummen, transportiert werden.
Des Weiteren wird auf Anwendungsebene ebenfalls eine *MTU* definiert. Darüber wird definiert wie groß aus Sicht der Anwendung eine Nachricht sein darf. Diese Nachrichtengröße wird beispielsweise für zusätzliche Überprüfungen der ausgetauschten Daten verwendet. Sie hat allerdings keinen Einfluss auf die Größe der versendeten Pakete.
Vor allem der Wert der *MTU* auf Sicherungsschicht hat nachhaltigen Einfluss auf die Performance des Netzwerkes. Die optimale Einstellung ist je nach Verwendungszweck, aber auch nach eingesetzter Hard- und Software unterschiedlich.

Zur weiteren Verbesserung der Performance von Infiniband wird der Großteil der Kommunikation mittels *Remote Direct Memory Access (RDMA)*, einer hochperformante Kommunikationstechnik, durchgeführt. Das heißt, dass durch viele *Verbs* eben diese Technik automatisch verwendet wird, ohne dass der Nutzer explizite Angaben vornehmen muss. Durch *RDMA* wird ermöglicht, dass durch Anwendungen lesend oder schreibend direkt auf entfernten Speicher zugegriffen werden kann. Der Adressbereich der Anwendung wird dazu mittels einer *QP* virtualisiert und beim eigenen *HCA* mit genauer Speicheradresse und einem Zugriffsschlüssel registriert.

Die gängigsten Lese- und Schreibzugriffe über *RDMA*, die bei Aufruf der entsprechenden *Verbs* ausgeführt werden, werden im Folgenden genauer betrachtet. Wird auf einem Host ein *RDMA-Send* aufgerufen, wird ein Workrequest an die *Send-Queue* des *QP* gestellt. Darüber wird die Speicherstelle der zu versendenden

Daten, sowie die Adresse des Empfängers spezifiziert. Daraufhin wird damit begonnen die Daten von der angegeben Speicherstelle an den Empfänger zu senden. Damit die Daten empfangen werden können, muss durch den Empfänger zuvor ein entsprechendes *RDMA-Receive* aufgerufen worden sein. Dadurch wird ebenfalls ein *Workrequest*, ein sogenannter *Receive-Request*, an die *Receive-Queue* des Empfängers gestellt. Darüber wird die Speicheradresse für die zu empfangenden Daten spezifiziert. Dadurch werden die Daten bei Eingang sofort an die richtige Stelle geschrieben. Der zeitlicher Ablauf des *RDMA-Sends* ist zur Verdeutlichung in Abbildung 3.2 dargestellt. Darin wird durch den Empfänger frühzeitig

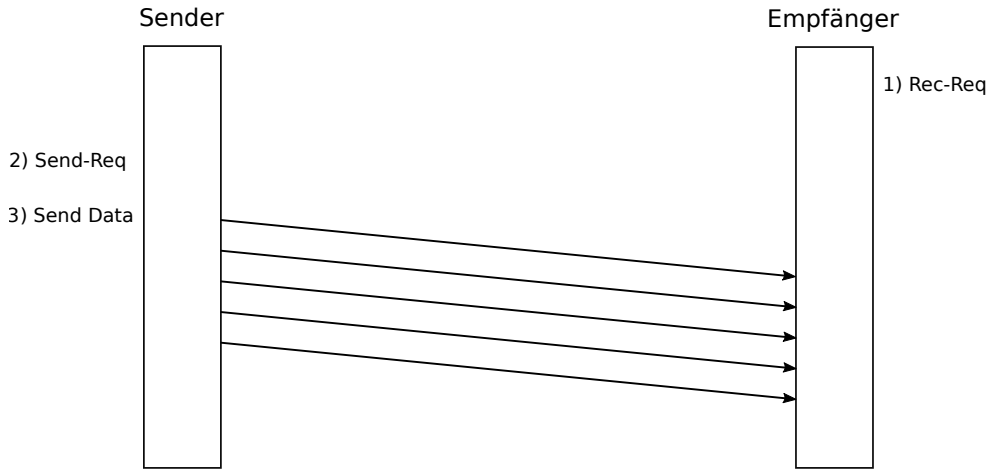

Abbildung 3.2: Zeitlicher Ablauf des *RDMA-Send*

ein *Receive-Request (Recv-Req)* aufgerufen. Dadurch wird ein Bereich im Hauptspeicher des Empfängers für die im Folgenden eingehenden Daten reserviert. Nachdem auf Senderseite die zu sendenden Daten über ein *Send-Request (Send-Req)* referenziert wurden, werden die Daten an den Empfänger verschickt. Auf Empfängerseite werden die Daten anhand der Informationen im *Receive-Request* direkt an die richtige Stelle des Hauptspeichers geschrieben.
Wird vom Sender ein *RDMA-Write* verwendet, ist kein explizites *RDMA-Receive* auf Empfängerseite notwendig. Stattdessen muss der Austausch des Zugriffsschlüssels und der entsprechenden Speicheradresse des Empfängers im Vorhinein auf Anwendungsebene erledigt werden. Bei Aufruf eines *RDMA-Writes*, wird wiederum ein Workrequest erstellt. Der *Write-Request* spezifiziert neben der lokalen Speicheradresse und der Adresse des Empfängers, auch die Speicherstelle des Empfängers und den benötigten Zugriffsschlüssel. Somit liegen alle notwendigen Informationen für den entfernten Speicherzugriff vor. Daher wird damit begonnen die entsprechenden Daten zu versenden. Auf Empfängerseite werden die Daten in einer gesonderten Queue empfangen und direkt an die entsprechende Speicherstelle geschrieben.
Sollen Daten von einem anderen Host gelesen werden, wird ein *RDMA-Read* verwendet. Dadurch wird ein entsprechender *Read-Request* erstellt. Darüber wird die

Speicherstelle, an die die Daten geschrieben werden sollen und die Speicherstelle, an der die zu lesenden Daten stehen, sowie der benötigte Zugriffsschlüssel für den entfernten Speicher referenziert. Diese Daten werden mittels eines *Read-Request-Packeges* an den anderen Host geschickt. Dort werden nach Überprüfung der Zugriffsrechte, die Daten versendet. Auf Empfängerseite werden die Daten an die Speicherstelle geschrieben, die über den zuvor erstellten *Read-Request* referenziert wird.
Alle Operationen, die für das Senden und Empfangen der Daten benötigt werden, erfolgen ohne Interaktion mit dem jeweiligen Betriebssystem. Dadurch wird zusätzlich auf allen beteiligten Hosts Rechenzeit eingespart, die der eigentlichen Anwendung zu Gute kommt.

Zur Verbesserung der Auslastung des Netzes werden auch eine Fluss- und eine Staukontrolle implementiert. Bei der Flusskontrolle wird auf Linkebene gearbeitet. Das heißt, dass in jeder Komponente eine Überprüfung stattfindet, bevor ein Paket über einen Link weitergeleitet wird. Dadurch wird sichergestellt, dass Pakete nur versendet werden, wenn der Empfänger auf der anderen Seite des Links genügend freie Pufferkapazität besitzt um das komplette Paket aufzunehmen. In einem Infiniband-Netzwerk können daher keine Pakete verloren gehen, sofern keine elektrischen Störungen auftreten. Als Basis für die Berechnung der freien Kapazität wird die totale Anzahl an Daten verwendet, die seit Initialisierung über den Link übertragen wurden. Von der empfangenden und der sendenden Seite wird dafür jeweils ein 12-Bit Zähler genutzt, in dem die Anzahl an gesendeten 64 Byte Blöcken gespeichert wird. Bei einem Überlauf wird erneut bei null begonnen zu zählen. Der aktuelle Wert wird zur Synchronisation fortlaufend durch den Sender an den Empfänger geschickt. Dafür wird das Feld *Flow Control Total Bytes Send (FCTBS)* in den sogenannten *Flow Control Packages* verwendet.
Auf Empfängerseite wird aufgrund des *FCTBS* das neue *Flow Control Credit Limit (FCCL)* berechnet. Dafür wird ein zusätzlicher Wert, die *Adjusted Blocks Received (ARB)* genutzt. Wird ein *Flow Control Package* empfangen, wird die *ARB* auf den Wert des *FCTBS* gesetzt. Bei Empfang eines Datenpaketes wird die Anzahl empfangener Blöcke zur *ARB* addiert und anschließend modulo 4096 gerechnet. Der Wert 4096 entspricht der maximalen Größe eines 12-Bit Zählers, wodurch Überläufe der Zähler verhindert werden. Das neue *FCCL* ist die Summe aus *ARB* und der Größe des freien Empfangspuffers in 64 Byte Blöcken, gefolgt von einer Modulorechnung mit 4096. Maximal werden 2048 freie Blöcke verwendet. Das *FCCL* wird regelmäßig und immer wenn benötigt an den Sender geschickt.
Auf Senderseite wird anhand des *FCCL* berechnet, ob ein Paket gesendet werden darf. Dazu wird vom *FCCL* die Summe aus dem aktuellen *FCTBS* und der Paketgröße abgezogen. Ist das Ergebnis modulo 4096 kleiner oder gleich 2048 kann ein Paket gesendet werden. Aufgrund der Modulorechnung werden auch hier etwaige Fehler bezüglich des Überlaufs der 12-Bit Zähler ausgeschlossen.

Aufgrund des festen Limits und der kontinuierlichen Synchronisierung wird sichergestellt, dass kein Paket aufgrund eines Pufferüberlaufs verloren gehen kann.

Zusätzlich zur Flusskontrolle auf Linkebene wird bei entsprechender Konfiguration eine Ende-zu-Ende Flusskontrolle genutzt. Dabei werden dem Sender vom Empfänger Kredite für das Versenden von Daten übertragen. Dadurch wird die Sendeleistung den Ressourcen des Empfängers angepasst. Die Kredite werden bei Bestätigungen im Header eingetragen oder auch in separaten Nachrichten verschickt. Die Berechnung und Einhaltung der Kredite sind über die *Workrequests* innerhalb der *QPs* in den Sende- und Empfangsablauf integriert.

Im Unterschied zur vorgestellten Staukontrolle von *TCP* wird in Infiniband eine *Staukontrolle mit Unterstützung des Netzwerkes* implementiert. Eine ähnliche Funktionalität wurde im Nachhinein auch in *TCP* integriert.
Wird von einem Switch in einem Infiniband-Netzwerk eine Überlast an einem Port erkannt, so wechselt dieser in den *Congestion State*. Dadurch werden die Pakete, die über den identifizierten Port versendet werden, mittels einer *Forward Explicit Congestion Notification (FECN)* markiert. Dazu wird ein Bit im entsprechenden Header-Feld gesetzt. Je nach Konfiguration kann auch nur ein Teil der Pakete oder nur Pakete einer bestimmten Anwendung markiert werden. Wird von einem Host eine *FECN* empfangen, wird mit der Bestätigung oder einer separaten Nachricht eine *Backward Explicit Congestion Notification (BECN)* versendet. Dazu wird ebenfalls ein Bit im Header gesetzt. Mittels dieser Nachricht wird der Sender über die Überlastsituation unterrichtet. Als Reaktion auf die Nachricht wird die Senderate verringert. Dadurch wird dem Stau entgegengewirkt. Wird von dem Switch schließlich keine Überlast mehr erkannt, verlässt das Switch den *Congestion State* und es werden keine weiteren Pakete markiert. Erst nach einer gewissen Zeit wird die Senderate wieder schrittweise erhöht. Sowohl die Stärke der Verringerung als auch die der Erhöhung der Senderate können über Parameter modifiziert werden.

Damit alle Komponente eines Infiniband-Netzwerks eindeutig identifiziert werden können, wird die sogenannte *Globally Unique Identifier (GUID)* verwendet. Die *GUID* wird einer Komponente, wie eine MAC-Adresse, direkt durch den Hersteller der Hardware fest zugeordnet. Zur Adressierung wird jedem Port ein *Local Identifier (LID)* zugewiesen. Eine *LID* ist innerhalb eines Subnetzes eindeutig. Die Weiterleitung von Paketen in Switches erfolgt anhand von Forwarding-Tabellen, in denen die *LIDs* von Komponenten Ausgangs-Ports zugeordnet werden.

Die Erstellung der Forwarding-Tabellen, sowie die Verteilung der *LIDs* fällt unter die Aufgaben des *Subnet Manager (SM)*. Der *SM* ist eine zentrale Kontroll- und Konfigurationseinheit. Die Instanz kann entweder auf einem Switch oder auf einem Host im Netz installiert werden.

Während des regulären Betriebes besteht die Hauptaufgabe des *SM* darin, die Vorgänge im Netz zu überwachen. Wird beispielsweise der Ausfall eines Links festgestellt, werden die Forwarding-Tabellen neu berechnet und an die Switches verteilt. Werden vermehrt Fehler an einer Komponente erkannt, wird durch den *SM* gegebenenfalls ein Port oder auch eine ganze Komponente deaktiviert.
Werden neue Komponenten dem Netz hinzugefügt, werden diese durch den *SM* registriert und mit einer *LID* versehen. Wenn möglich wird dabei auf alte Konfigurationen zurückgegriffen, damit im Netz möglichst wenige Änderungen vorgenommen werden müssen.
Durch die separate Instanz des *SM* wird eine Gesamtsicht über das Netz gewonnen. Dadurch werden effiziente Forwarding-Algorithmen und eine Bewertung des Zustandes des Netzes ermöglicht.
Aufgrund der Gesamtsicht wird eine weitere Funktion, die Definition von einzelnen Partitionen, an einer zentralen Stelle ermöglicht. Partitionen sind logische Trennungen des Netzes, die sich vor allem bei Nutzung unterschiedlicher Anwendungen auf dem selben Netz anbieten. Sie sind dabei unabhängig von Netzwerkkomponenten oder Subnetzen und basieren auf einem eigenen *IP-Adressraum*. Die Zugehörigkeit zu einer Partition wird über eine Liste von *GUIDs* definiert. Eine *GUID* kann auch mehreren unterschiedlichen Partitionen zugeordnet werden. Jede Partition wird über einen *Partition-Key* eindeutig referenziert, der bei Kommunikation innerhalb einer Partition in den Paket-Header eingetragen wird. Im ganzen Netz ist grundsätzlich eine *Default-Partition* mit allen *GUIDs* definiert.

Generell wird zur Kommunikation von allen Informationen zur Konfiguration und Steuerung der Komponenten eine separate Paketklasse, die *Management-Pakete*, verwendet. Sie unterscheiden sich in ihrem Aufbau grundlegend von den ansonsten verwendeten Daten-Paketen. Dadurch ist es auch möglich Pakete zuzustellen, wenn keine *LIDs* verteilt oder Forwarding-Tabellen fehlerhaft sind. Dazu wird ein direktes Forwarding mit einer Liste von aufeinander folgenden Ausgangs-Ports verwendet.

Im folgenden Kapitel 3.3 wird die umgesetzte Architektur von *QoS* auf Infiniband genauer betrachtet und in Kapitel 3.4 die genauen Konfigurationsmöglichkeiten aufgezeigt.

3.3 QoS Architektur

Infiniband bietet eine, im Vergleich zu den in Abschnitt 2.4.4 und 2.4.5 vorgestellten *QoS-Architekturen*, eher einfache Struktur. Dies liegt vor allem daran, dass die Unterstützung von *QoS* im Design von Infiniband berücksichtigt und nicht wie bei *IP* nachträglich hinzugefügt wurde [10].
In Infiniband wird im Grunde eine leichte Variation von *DiffServ* implementiert. Der Hauptunterschied ist, dass in Infiniband-Netzwerken *QoS* auf Basis der Switches umgesetzt wird. Es wird demnach nur im eigenen Netz gearbeitet. Prinzipiell sind Router im Infiniband-Standard definiert, werden aber bisher von keinem Anbieter hergestellt.

Auch für Infiniband bilden die in Abschnitt 2.4.3 erarbeiteten Prinzipien *Klassifizierung*, *Isolation von Scheduling und Policing*, sowie *Zugangssteuerung* die Basis zur Umsetzung von *QoS*.

Zur *Klassifizierung* müssen die Pakete mit unterschiedlichen Markierungen versehen werden. Die Markierung der Datenströme und somit auch die Zuordnung in unterschiedliche Klassen erfolgt mittels der sogenannten *Service Levels (SLs)*. Das verwendete *SL* wird, analog zum *DSCP* bei *IP-Paketen*, in jedem Paket ins entsprechende Header-Feld eingetragen. Im Standard werden 16 verschiedene *SLs* definiert, somit können 16 unterschiedliche Prioritätsklassen verwendet werden. Die Bestimmung des *SL* eines Paketes erfolgt aufgrund von unterschiedlichen Informationen. Beispielsweise kann das *SL* durch das verwendeten *ULP* oder durch die genutzten Partition bestimmt werden. Bei der Konfiguration des *SL* können auch Parameter für das *Policing* des Datenstroms angegeben werden. Auch die *Zugangssteuerung* erfolgt implizit über die Zuweisung der *SL*. Allen Datenströmen wird statisch ein *SL* zugeordnet. Für alle nicht explizit konfigurierten Datenströme wird ein *Default-SL* definiert. Da nur innerhalb des eigenen Netzes gearbeitet wird, werden keine weiteren Mechanismen benötigt. Eine Einteilung entsprechend der *Kernnetz-* beziehungsweise der *Zugangs-Router* in *DiffServ* wird daher nicht benötigt. Aufgrund dessen wird auch die zusätzliche Kontrolleinheit, der *Bandwidth Broker*, für die Umsetzung von *QoS* auf Infiniband nicht benötigt.
Die Konfiguration der Zuweisung der *SL* und die Definition des *Policing* werden in Abschnitt 3.4.2 genauer beschrieben.

Die *Isolation des Schedulings* wird auf den Switches durch die separate Behandlung der *SL* umgesetzt. Dazu werden pro physischem Link bis zu 16 *Virtual Lanes (VLs)* definiert. Die genaue Anzahl der zur Verfügung stehenden *VLs* hängt von der verwendeten Hardware, sowie der installierten Treiberversion ab. Generell können *VLs* als virtuelle Kanäle, die einen physischen Link im mehrere logische Einheiten aufteilen, beschrieben werden. Dabei wird jeder *VL* ein separater Speicherbereich zugeordnet.

Alle *SLs* werden auf *VLs* abgebildet. Die Zuweisung der *SLs* auf die *VLs* wird über eine statische Konfiguration des *SM* vorgenommen. Dies wird auch *Service Level to Virtual Lane Mapping* genannt. Eine genaue Beschreibung der Konfigurationsmöglichkeiten des *SL to VL Mappings* wird in Abschnitt 3.4.1 vorgenommen.
Die Fluss- und Staukontrolle wird ebenfalls vom physischen Link abstrahiert und separat auf die einzelnen *VLs* angewendet. Insgesamt wird dadurch eine komplett separate Behandlung einzelner *SLs* ermöglicht.
Ein Ausnahme davon bildet die *VL 15*, welche nur für Managementdaten verwendet wird. Generell werden Management-Pakete immer vor allen anderen Paketen versendet. Sie unterliegen außerdem keiner Stau- und Flusskontrolle. Allerdings ist der Umfang des Management-Verkehrs im normalen Betrieb sehr niedrig, sodass die Pakete der Flusskontrolle und die Datenpakete in der Regel nicht beeinflusst werden.

Die *VLs* können auf den Switches mittels eines *Dual Priority WFQ* beliebig priorisiert werden. Dafür werden zunächst die *VLs* in zwei Klassen, eine mit hoher und eine mit niedriger Priorität, eingeteilt. Innerhalb dieser Klassen wird jeweils ein *WFQ* angewendet. Dazu wird jeder *VL* ein Gewicht zugeordnet und dementsprechend ein Prozentsatz der Leistung des Switches garantiert. Die Garantie gilt wie bei *DiffServ* nur, wenn die Verkehrsprofile keine Überlast innerhalb einer *VL* erzeugen. Die genaue Priorisierung der *VLs*, sowie die daraus entstehenden Garantien werden in Abschnitt 3.4.1 behandelt.
Grundsätzlich gilt, dass alle Pakete der *VLs* mit hoher Priorität immer vor den Paketen mit niedriger Priorität gesendet werden. Eine Ausnahme davon wird durch den Parameter *Limit of High-Priority* realisiert. Durch den Wert wird festgelegt, nach wie vielen hoch priorisierten Paketen ein Paket mit niedriger Priorität verschickt wird. Dies dient dazu, dass der niedrig priorisierte Verkehr nicht komplett abgeschnitten wird. Alle Einstellungen bezüglich der Nutzung eines physischen Links werden unter dem Begriff *VL Arbitration* zusammengefasst. Ein Eingriff in die Priorisierung der Pakete über unterschiedliche *Discard Policies* kann aufgrund der verlustfreien Flusskontrolle nicht vorgenommen werden.

In Abbildung 3.3 ist zur Veranschaulichung ein Link mit vier *VLs* dargestellt. Die Pakete sind farblich entsprechend der genutzten *VL* markiert. Die Zuordnung zu den einzelnen *VLs* erfolgt aufgrund der *SLs* der Pakete. Sie wird im Beispiel nicht dargestellt. Es wird angenommen, dass durch die *VL Arbitration* des Beispiels 50 % der zur Verfügung stehenden Bandbreite *VL 0* zugeordnet wird. Die anderen *VLs* werden mit gleicher Priorität behandelt.
Daraus resultiert, dass jedes zweite Paket, das über den physischen Link geschickt wird, entsprechend der *VL 0 gelb* markiert ist. Die Pakete dazwischen werden gleichermaßen von *VL 1* (*rot*) und *VL 2* (*blau*) eingenommen. Da der Empfangspuffer für *VL 3* voll ist, wird durch die Flusskontrolle verhindert, dass ein *grünes* Paket gesendet wird.

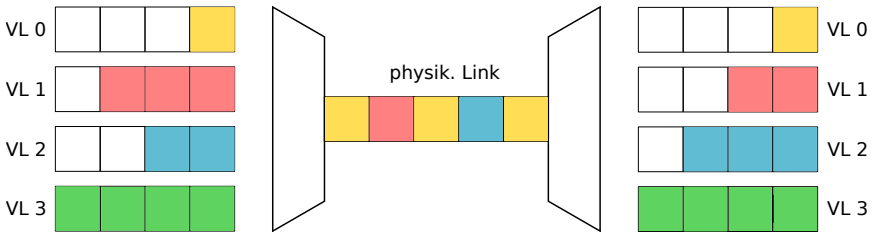

Abbildung 3.3: Aufteilung eines Links in *Virtual Lanes*

Damit die *QoS-Mechanismen* wie in Abbildung 3.3 dargestellt greifen, muss auf dem Link eine Überlastsituation vorliegen. Das heißt, dass Pakete drauf warten müssen gesendet zu werden, weil die Kapazität des Links komplett verwendet wird.
In der Regel treten solche Situationen an besonders stark belasteten Punkten eines Netzwerkes auf. Oft stehen auch Links unter erhöhter Last, wenn durch mehrere Knoten Daten an einen einzelnen Knoten geschickt, beziehungsweise von einem Knoten gelesen werden werden.

Wie auch bei *DiffServ* können bei *QoS* auf Infiniband keine harten Dienstgarantien ausgesprochen werden, da für eine Verbindung kein separater Kanal verwendet wird. Bei Einhaltung von vorgegebenen Verkehrsprofilen, beziehungsweise bei Betrachtung einzelner *SLs*, können konkrete Aussagen über die zur Verfügung stehende Bandbreite beziehungsweise Latenz getroffen werden.

Die genaue Konfiguration von *QoS* auf Infiniband wird im folgenden Kapitel 3.4 behandelt. Zunächst wird in Abschnitt 3.4.1 das *SL to VL Mapping* und die Konfiguration des *Dual Priority WFQ* betrachtet. In Abschnitt 3.4.2 werden die unterschiedlichen Möglichkeiten zur Zuweisung der *SL* dargelegt. Für den erfolgreichen Einsatz von *QoS* auf Infiniband müssen die beiden separaten Konfigurationen in einem in sich stimmigen Konzept erarbeitet werden.

3.4 QoS Konfiguration

3.4.1 SL to VL Mapping und VL Arbitration

Die Konfiguration des *SL to VL Mappings* und die der *VL Arbitration* werden im *SM* vorgenommen. Im Folgenden wird die *SM-Implementierung* von *OpenSM* verwendet. Dabei werden alle zentralen Konfigurationen des *SM* in der Datei *opensm.conf* vorgenommen.

Für das Mapping werden zwei Werte verwendet. Der Parameter *qos_max_vls* beschränkt die Anzahl der verwendeten *VLs*. Maximal können 16 *VLs* angegeben werden. Das Mapping erfolgt über den Parameter *qos_sl2vl*, dem eine Liste mit Integer-Werten zugewiesen wird. Über die Liste werden die *VLs* und die *SLs* miteinander verknüpft. Die Werte der Liste entsprechen der jeweiligen *VL*. Die *SLs* werden über die Position, an der die Werte in der Liste stehen, referenziert. Wird einem *SL* die *VL 15* zugeordnet, wird dieser blockiert, da *VL 15* nur für Management-Verkehr zugelassen ist.

Die *VL Arbitration* wird über den Parameter *qos_high_limit*, sowie die Listen *qos_vlarb_high* und *qos_vlarb_low* konfiguriert.
Der Wert des Parameters *qos_high_limit* gibt an, wie viele Pakete aus der hoch priorisierten Warteschlange gesendet werden dürfen, bis ein Paket aus der niedrig priorisierten gesendet werden muss. Der Wert muss zwischen einschließlich null und 255 liegen. Ein Wert von null bedeutet, dass nur genau ein Paket gesendet werden darf. Bei einem Wert von 255 wird nur ein niedrig priorisiertes Paket gesendet, wenn kein hoch priorisiertes in der Warteschlange ist. Ansonsten werden maximal so viele Daten gesendet bis die Schranke, aus dem Produkt des Wertes des Parameters und 4096 Bytes, erreicht wird.
In der Liste des Parameters *qos_vlarb_high* werden die hoch priorisierten *VLs* definiert. Dazu werden Paare von *VLs* und dem dazu gehörigen Gewicht eingetragen. Die Gewichte müssen ebenfalls einen Wert zwischen einschließlich null und 255 haben. Ist die entsprechende *VL* an der Reihe, können so viele Daten gesendet werden, bis die Schranke aus dem Produkt des Gewichtes und 64 Bytes erreicht wird. Hat ein Gewicht den Wert null wird diese *VL* übersprungen.
Die *VLs* werden nach der Reihenfolge in der Liste und entsprechend ihrer Gewichte abgearbeitet. In der Liste kann die gleiche *VL* mehrmals vorkommen. Dadurch kann Einfluss auf die maximale Latenz, sowie die Größe von *Bursts* in einzelnen *VLs* genommen werden.
Die Konfiguration der Liste *qos_vlarb_low* erfolgt analog. Es ist auch möglich, dass *VLs* sowohl in der hoch als auch in der niedrig priorisierten Liste vorkommen. Dadurch werden komplexere Konfigurationen ermöglicht.

Zur Auswahl des als nächstes zu sendenden Paketes wird wie folgt vorgegangen. Zunächst wird bestimmt, welche Liste an der Reihe ist. Dazu wird kontrolliert,

ob in einer der *VLs* mit einem Gewicht größer als null der hoch priorisierten Liste ein Paket zum Versenden vorliegt. Ebenfalls muss ein Senden laut Flusskontrolle innerhalb dieser *VL* möglich sein. Wird kein entsprechendes Paket gefunden, wird direkt mit der niedrig priorisierten Liste fortgefahren.
Ansonsten wird der sogenannte *HighPriCounter* kontrolliert. Der *HighPriCounter* wird immer, wenn von der hoch auf die niedrig priorisierte Liste gewechselt wird, auf das Produkt aus dem Wert des *qos_high_limit* und 4096 gesetzt. Ist der Wert des Zählers negativ, wird ebenfalls direkt zur niedrig priorisierten Liste gewechselt. Andernfalls wird das zuvor gefundene Paket gesendet. Wurde das *qos_high_limit* auf einen kleineren Wert als 255 konfiguriert, wird die Länge des Paketes vom *HighPriCounter* abgezogen.
Innerhalb der beiden Listen wird jeweils ein *WFQ* angewendet. Die Reihenfolge der Abarbeitung entspricht der der jeweiligen Liste. Für jede der beiden Listen wird ein Zeiger und eine Variable ω angelegt. Der Zeiger referenziert das aktuelle Element der Liste und ω die maximal zu versendende Datenmenge. Der Wert für ω ergibt sich aus dem Produkt des in der Liste angegebenen Gewichtes und 64 Byte.
Ein Paket der aktuellen *VL* wird gesendet, wenn ω positiv ist, ein Paket in der Warteschlange ist und keine Blockade durch die Flusskontrolle vorliegen. Wurde ein Paket gesendet, wird die Länge des Paketes von ω abgezogen. Darf beziehungsweise kann aus der *VL* kein Paket gesendet werden, wird der Zeiger auf den nächsten Eintrag in der Liste verschoben und ω mit dem zugehörigen Gewicht neu berechnet. Nach diesem Prinzip wird sukzessiv fortgefahren.
Immer wenn ein Paket gesendet wurde, wird erneut geprüft welche Liste als nächstes bearbeitet wird. Durch einen Wechsel der Listen werden die internen Zeiger und Zähler nicht verändert, damit nach einem erneuten Wechsel an der gleichen Stelle fortgefahren werden kann.
Zur Veranschaulichung ist in Abbildung 3.4 eine einfache Konfiguration angegeben. Es werden maximal 4 *VLs* zugelassen und ein Limit von 100 gesetzt. Das

```
#QoS options

qos_max_vls 4
qos_high_limit 100

qos_vlarb_high 1:48,2:64,1:48,3:32
qos_vlarb_low 0:100,1:0,2:0,3:0

qos_sl2vl 0,1,2,3,0,0,0,0,0,0,0,0,0,0,0,0
```

Abbildung 3.4: Beispielhafte Konfiguration der *QoS-Optionen* in *opensm.conf*

heißt, dass spätestens nachdem 400 KB Daten aus der hoch priorisierten Liste gesendet wurden, ein Paket aus der niedrig priorisierten Liste verschickt werden kann.

Im Beispiel ist nur *VL 0* niedrig priorisiert. Alle anderen *VLs* sind, wie im Standard empfohlen, mit einem Gewicht von null in der Liste aufgeführt. Dadurch hat auch das Gewicht von *VL 0* keinen weiteren Einfluss. Die *VLs 1-3* sind hoch priorisiert. Anhand der Gewichte wird *VL 1* die Hälfte, *VL 2* ein Drittel und *VL 3* ein Sechstel der Bandbreite zugesichert. Durch die Aufteilung des Gewichtes von *VL 1* in zwei Einträge wird erreicht, dass die Zeit zwischen der Weiterleitung der Pakete kleiner und auch konstanter ist. Das heißt, dass dadurch Latenz und Jitter kontrolliert werden können. Außerdem wird auch der Burst aus *VL 1* begrenzt, allerdings nur dann, wenn auch wirklich Pakete aus *VL 2* beziehungsweise *VL 3* gesendet werden.
Anhand der Liste des Parameters *qos_sl2vl* werden die *SL 0-3* entsprechend den *VLs 0-3* zugeordnet. Alle Pakete mit einem anderen *SL* werden ebenfalls über *VL 0* verschickt.

Im Folgenden wird die Auswirkung der Konfiguration für *VL 1* betrachtet. Es wird davon ausgegangen, dass in jeder *VL* immer ausreichend Pakete vorhanden sind und keine Blockade durch die Flusskontrolle auftritt. Außerdem wird zur Vereinfachung angenommen, dass alle Pakete eine Größe von 2048 Byte haben.
Nach dem oben vorgestellten Algorithmus wird zunächst die hoch priorisierte Liste ausgewählt und beim ersten Eintrag begonnen. Die maximale Datenmenge ω wird auf 3072 (48×64) gesetzt. Nach dem Senden des ersten Paketes hat ω einen Wert von 1024. Da der Wert positiv ist, wird ein weiteres Paket gesendet. Daraufhin wird ω entsprechend auf -1024 reduziert.
Aufgrund des negativen Wertes wird mit dem nächsten Eintrag der Liste fortgefahren. Aus *VL 2* werden ebenfalls zwei Pakete versendet. Dann ist nochmals *VL 1* an der Reihe. Wegen des gleichen Gewichtes wie beim ersten Eintrag werden wieder zwei Pakete verschickt. Schließlich wird ein Paket aus *VL 3* versendet. Damit ist die hoch priorisierte Liste einmal abgearbeitet. Durch das *qos_high_limit* von 100, wird erst nachdem 200 Pakete aus der hoch priorisierten Liste gesendet wurden, genau eins aus der niedrig priorisierten Liste versendet. Das bedeutet, dass im schlechtesten Fall drei Pakete anderer *VLs* zwischen zweien aus *VL 1* liegen können.

3.4.2 Zuweisung unterschiedlicher Service Level

Die Konfiguration der Zuweisung der *SL* wird in der Datei vorgenommen, die in der *opensm.conf* unter dem Parameter *qos_policy_file* dem *SM* bekannt gegeben wird. Für die Konfiguration stehen die *Advanced-* und die *Simple-Syntax* zur Verfügung. Logischerweise bietet die *Advanced-Syntax* deutlich mehr Einstellungsmöglichkeiten. Es ist auch möglich beide Syntaxarten innerhalb einer Konfigurationsdatei zu kombinieren.

Die *Simple-Syntax* besteht aus einer Liste von Regeln, die bei Verwendung eines *ULP* der darauf zugreifenden Anwendung ein *SL* zuweist. Eine *Default-Regelung*

ist dabei zwingend erforderlich. Je nach *ULP* können unterschiedliche Kriterien zur genaueren Abgrenzung der Regel verwendet werden. Beispielsweise können je nach verwendeter Portnummer unterschiedliche *SLs* zugewiesen werden. Weitere Einschränkungen existieren nicht. Bei der Bestimmung des *SL* für einen Datenstrom werden die Regeln nach der Reihenfolge in der Konfigurationsdatei abgearbeitet. Die erste zutreffende Regel wird angewendet. Daher muss auf die gewünschte Reihenfolge geachtet werden, da Überlappungen in den Geltungsbereichen der Regeln möglich sind. Wird keine zutreffende Regel gefunden wird der *Default-Eintrag* verwendet.
In Abbildung 3.5 ist eine minimale Konfiguration zur Veranschaulichung angegeben. Zwischen den Schlüsselwörtern *qos-ulps* und *end-qos-ulps*, die Anfang und

```
qos-ulps
   default : 0
end-qos-ulps
```

Abbildung 3.5: Minimalbeispiel einer Konfiguration mit der *Simple-Syntax*

Ende der Datei markieren, wird nur das *Default-SL* auf das *SL 0* gesetzt. Somit wird jeglicher Verkehr mit dem *SL 0* versehen. Eine Ausnahme davon bilden hardwarenahe Kommandos, bei denen das *SL* mittels eines Kommandozeilenparameters gesetzt werden kann. Eine weitere Ausnahme ist *MPI*, bei dem das zu verwendende *SL* ebenfalls explizit gesetzt werden kann.
Eine umfangreichere Konfiguration ist in Abbildung 3.6 dargestellt. Zunächst wird das *Default-SL* auf das *SL 0* gesetzt. Dann folgen zwei Einträge für das *SDP*.

```
qos-ulps
   default                       : 0
   sdp, port-num 20000-30000 : 1
   sdp                           : 2
   ipoib, pkey 0x0001        : 3
   ipoib                         : 2
   any, pkey 0x1234          : 4
   any, target-port-guid 0xABC0-0xABCF : 5
end-qos-ulps
```

Abbildung 3.6: Umfangreiche Konfiguration mit der *Simple-Syntax*

Im ersten Eintrag wird zur genaueren Abgrenzung ein Portintervall von 10000 bis 20000 angegeben und das *SL 1* zugewiesen. Werden andere Ports verwendet trifft erst der zweite Eintrag zu und es wird das *SL 2* verwendet. Durch die nächsten beiden Einträge wird *IPoIB* in zwei unterschiedliche *SL* aufgeteilt. Wird innerhalb einer bestimmten *Partition* über *IPoIB* kommuniziert, wird ein anderes *SL* verwendet. Im Beispiel wird bei *IPoIB* innerhalb der *Partition 0x0001* das *SL 3* verwendet, ansonsten *SL 2*. Es ist auch möglich, dass *SLs* unabhängig

des *ULP* zugeordnet werden. Dazu wird das Schlüsselwort *any* verwendet. In der abgebildeten Konfiguration wird bei Kommunikation innerhalb der *Partition 0x1234* das *SL 4* verwendet. Pakete, die an einen Port aus der *GUID-Gruppe* von *0xABC0* bis *0xABC0* adressiert sind nutzen das *SL 5*.

Für komplexere Zuweisungen, sowie für die Verwendung von *Policing*, muss die *Advanced-Syntax* verwendet werden. Eine typische Konfigurationsdatei lässt sich in die drei Bereiche *Port Groups*, *QoS Levels* und *QoS Matching Rules* aufteilen. Sie werden mit entsprechenden Schlüsselwörtern voneinander abgegrenzt. Im Abschnitt *Port Groups* können Gruppen definiert werden, die von den später definierten Regeln verwendet werden können. Kriterien zur Erstellung der Gruppen sind beispielsweise die *GUID* oder definierte Partitionen für *IPoIB*.
Im zweiten Bereich werden die unterschiedlichen *QoS-Level* definiert. Anders als in der *Simple-Syntax* kann nicht nur das *SL* definiert werden, sondern optional auch ein *MTU-Limit* und eine maximale Senderate angegeben werden. Als einzige Bedingung muss wieder ein *Default-Level* angelegt werden.
Innerhalb des Abschnitts *QoS Matching Rules* werden schließlich die Regeln zur Zuweisung definiert. Dafür können Sender und Empfänger jeweils als *Port Group* angegeben, sowie eine *Partition* und eine *Service-ID* definiert werden. Damit eine Regel erfüllt wird, muss ein Datenstrom allen angegebenen Werten entsprechen. Nur dann wird das *SL* den Paketen zugeordnet. Wird keine Regel erfüllt wird der *Default* verwendet.
Wie auch in der *Simple-Syntax* werden die Definitionen und Regeln der Reihe nach abgearbeitet und bei der ersten zutreffenden Regel die Eingruppierung vorgenommen.
Außerhalb der drei Bereiche der *Advanced-Syntax*, können in der selben Datei zusätzlich Definitionen in der *Simple-Syntax* vorgenommen werden. Je nach Reihenfolge der Konfigurationen wird der Bereich vor oder nach den Regeln der *Advanced-Syntax* abgearbeitet

Eine beispielhafte Konfiguration in der *Advanced-Syntax* ist in Abbildung 3.7 dargestellt. Darin werden zwei *Port Groups* definiert. Eine Storage-Einheit wird anhand ihrer *GUIDs* und eine *Partition* mittels ihres *Partition-Keys* referenziert. Für die differenzierte Behandlung der Verbindungen werden drei unterschiedliche *QoS Levels* eingerichtet. Diese umfassen das *Default-Level*, das auf *SL 0* abgebildet wird, sowie ein hoch und ein niedrig priorisiertes Level. Bei der niedrigen Priorisierung wird neben der Zuweisung des *SL 2* auch die Senderate beschränkt. Dafür wird der Parameter *rate-limit* auf 2 gesetzt. Darüber wird eine maximale Senderate von 2.5 Gb/s festgelegt. Dem hoch priorisierten Level *Prio* wird *SL 1* zugewiesen.
Im dritten Abschnitt werden schließlich die tatsächlichen Zuweisungsregeln definiert. Einerseits wird jeglicher Datenverkehr, der als Zieladresse einen Port aus dem oben definierten Storage hat und einen bestimmten Service nutzt, dem hoch

```
port-groups
   port-group
     name: Storage
     port-guid: 0x10001, 0x10005-0x10009
   end-port-group

   port-group
     name: Partitions
     partition: Part1
     pkey: 0x1234
   end-port-group
end-port-groups

qos-levels
   qos-level
     name: DEFAULT
     sl: 0
   end-qos-level

   qos-level
     name: Prio
     sl: 1
   end-qos-level

   qos-level
     name: Low
     sl: 2
     rate-limit: 2
   end-qos-level
end-qos-levels

qos-match-rules
   qos-match-rule
     destination: Storage
     service-id: 0x100001
     qos-level-name: Prio
   end-qos-match-rule

   qos-match-rule
     source: Part1
     qos-level-name: Low
   end-qos-match-rule
end-qos-match-rules
```

Abbildung 3.7: Konfiguration mit der *Advanced-Syntax*

priorisierten Level zugeordnet. Andererseits wird aller Verkehr, mit einem Sender aus der definierten Partition, niedrig priorisiert.
Der restlich anfallende Datenverkehr fällt unter den *Default* und wird mit dem *SL 0* markiert.

Durch diese beispielhafte Konfiguration werden bei entsprechender *VL Arbitration* zwei Ziele realisiert. Es wird eine Storage-Einheit definiert deren Zugriff über bestimmte Protokolle priorisiert wird. Außerdem wird der Datenverkehr innerhalb einer bestimmten Partition separiert und in der Senderate limitiert. Somit wird die Partition niedriger Eingestuft als der *Default-Verkehr*.

4 Umsetzung von Quality of Service auf Infiniband

4.1 Diskussion neuer Möglichkeiten

Für den gezielten Einsatz von *QoS* auf Infiniband muss die Konfiguration auf die vorliegenden Anforderungen angepasst werden. In diesem Kapitel werden dazu zwei der häufigsten Anwendungsfälle von Infiniband diskutiert. Zunächst wird auf die Anforderungen eines Clusters an das Netzwerk eingegangen. Im weiteren Verlauf wird die *Backbone-Struktur* eines Datacenters betrachtet.

Die Rechenleistungen des betrachteten Clusters soll für mehrere Nutzergruppen zur Verfügung gestellt werden. Durch die Administration wird demnach die Rolle eines Dienstleisters eingenommen. Daher muss mittels der Netzwerkstruktur der reibungslose Betrieb des Clusters gesichert werden. Außerdem soll dem Nutzer eine möglichst gute Performance geliefert werden.
Aufgrund der Erfahrungen im *Jülich Supercomputing Centre* mit Clustern wird angenommen, dass der Großteil der Netzwerklast durch Nutzeranwendungen auf Basis von *MPI*, sowie durch Storage-Zugriffe erzeugt wird. Durch Anwendungen auf Basis von *IPoIB* wird nur ein geringer Prozentsatz der Kapazität beansprucht.
Jeglicher Verkehr durch Administrations- und Managementwerkzeuge, wie zum Beispiel dem Batch-System, wird auf ein separates Ethernet-Netzwerk ausgelagert. Dass heißt, dass dadurch die Datenströme auf dem Infiniband-Netzwerk nicht beeinflusst werden.
Durch *QoS* soll eine genauere Einteilung der Netzwerklast erfolgen, damit dem Nutzer gegenüber bessere Performance und somit ein besseres Produkt geliefert werden kann. Außerdem kann darüber eine Anpassung des Clusters an die Anwendungen, aber auch eine betriebliche Ausrichtung realisiert werden.
Ziel der Nutzung von *QoS* ist es eine Konfiguration zu finden, in der die Kommunikation mit *MPI*, möglichst wenig durch anderen Verkehr beeinträchtigt wird. Des Weiteren soll die Storage-Anbindung separiert werden, damit darauf Einfluss genommen werden kann. Aufgrund der Verwendung des Filesystems *Lustre* soll eine weitere Unterscheidung zwischen dem *Storage Control Verkehr*, der niedrige Latenzen benötigt, und dem *Storage Data Verkehr*, der wiederum vermehrt Bandbreite benötigt, getroffen werden.

Generell wird der *MPI-Verkehr* als wichtigstes Gut im Cluster angesehen. Allerdings soll auch die Storage-Anbindung weiterhin den hohen Anforderungen eines Clusters entsprechen. Dass heißt, dass durch *MPI-Anwendungen* nicht das gesamte Netz blockiert werden darf. Außerdem soll eine Gruppe von Knoten im Cluster definiert werden, die mit besserer Performance als die restlichen Knoten an den Storage angebunden sind. Eine schematische Übersicht zu dem beschriebenen Cluster ist in Abbildung 4.1 dargestellt.

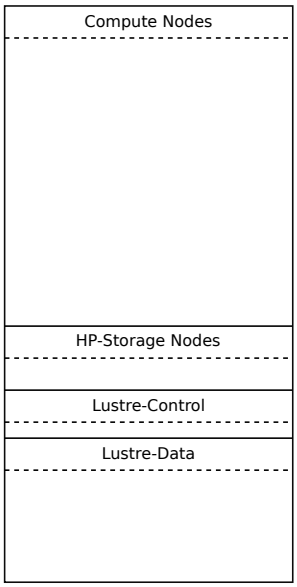

Abbildung 4.1: Übersicht des betrachteten Clusters

Zur Verdeutlichung der Auswirkungen, die der Einsatz von *QoS* auf dem beschriebenen Cluster mit sich bringt, ist in Abbildung 4.2 ein beispielhafter Verlauf der Zusammensetzung der Bandbreite dargestellt. Darin werden drei unterschiedliche Anwendungen betrachtet. Zu Beginn wird durch das hoch priorisierte *MPI* (*rot*)

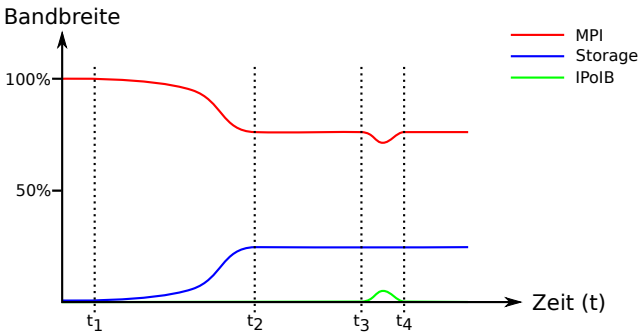

Abbildung 4.2: Zeitlicher Verlauf der Zusammensetzung der Bandbreite des betrachteten Clusters

die gesamte Bandbreite eingenommen, da durch keine weitere hoch priorisierte Anwendung kommuniziert wird. Zum Zeitpunkt t_1 beginnt ein Storage-Zugriff (*blau*), dem ebenfalls ein Teil der Bandbreite garantiert wird.
Bis zum Zeitpunkt t_2 steigt die durch den Storage-Zugriff genutzte Bandbreite auf die maximal garantierten 25 % an. Daher kann derweil durch *MPI* nur weniger Bandbreite beansprucht werden. Zwischen den Zeitpunkten t_3 und t_4 sinkt die durch *MPI* genutzte Bandbreite soweit ab, dass trotz des Storage-Zugriffs freie Kapazität entsteht. Diese freie Kapazität wird durch eine niedrig priorisierte Anwendung auf Basis von *IPoIB* genutzt. Sobald durch *MPI* wieder mehr Bandbreite benötigt wird, sinkt wiederum die für *IPoIB* zur Verfügung stehende Kapazität. Ab dem Zeitpunkt t_4 wird durch *MPI* und den Storage-Zugriff erneut die gesamte Bandbreite eingenommen, sodass durch die *IPoIB-Anwendung* keine Bandbreite mehr verwendet werden kann.

Als zweites Anwendungsbeispiel wird ein Datacenter mit unterschiedlichen Servern betrachtet. Die angebotenen Dienste umfassen ein Datenbank-System, ein Storage-System, eine Backup-Struktur, sowie einen Cloud-Server mit *virtuellen Maschinen (VMs)*. Die *VMs* werden als Workstations, sowie als zusätzliche Systeme (im Folgenden *Fat-Nodes* genannt) für aufwendigere und möglichst performante Berechnungen genutzt. Die einzelnen Server sind über ein Infiniband-Netzwerk zusammengeschlossen. Der Zugriff auf die Workstations und somit auch auf alle anderen Dienste, wird über Clients, die über ein Gateway an das Infiniband-Netzwerk angeschlossen sind, durchgeführt. Eine strukturelle Übersicht des Datacenters ist in Abbildung 4.3 dargestellt. Der Fokus für die Anwen-

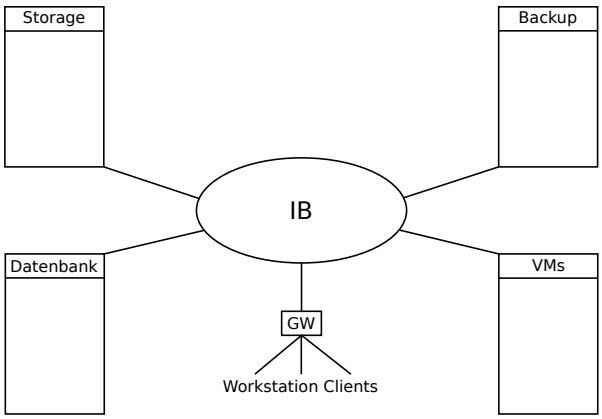

Abbildung 4.3: Übersicht des betrachteten Datacenters

dung von *QoS* wird auf die *Backbone-Struktur* gelegt, da hier mehrere unterschiedliche Dienste die selbe Hardware nutzen.
Durch die Anwendung von *QoS* können die einzelnen Verbindungen innerhalb des Netzes separiert und unterschiedlich behandelt werden. Dadurch wird das

Ziel verfolgt, die Performance der angebotenen Infrastruktur zu verbessern, indem Qualitätsgarantien entsprechend der Anforderungen der einzelnen Systeme erstellt werden. Dafür müssen die Dienste, wie in Kapitel 2.4.1 vorgestellt, bezüglich der benötigten Bandbreite und Latenz analysiert werden.
Die Anbindung des Backup-Systems soll so gestaltet werden, dass durch die Verbindungen möglichst kein anderer Verkehr behindert wird. Allerdings muss darauf geachtet werden, dass Backups auch bei voller Auslastung des Netzes nach einer gewissen Zeit abgeschlossen sein müssen. Ansonsten würde der Unterschied zwischen den aktuellen Daten und dem entsprechenden Backup immer mehr divergieren.
Der *VM-Server* wird für zwei unterschiedliche Aufgaben, die im Netz separat behandelt werden sollen, genutzt. Die *VMs* der Workstations benötigen eine geringe Latenz, damit auf ihnen ohne Verzögerung gearbeitet werden kann. Eine besonders hohe Bandbreite wird dabei nicht vorausgesetzt. Bei den *VMs* der *Fat-Nodes* wird ein anderes Verhalten benötigt. Sie sollen besonders priorisiert werden, da von ihnen aus performante Zugriffe auf das Datenbank-System, sowie auf den Storage erfolgen sollen.
Sowohl für das Storage-System, als auch das Datenbank-System wird vor allem eine hohe Bandbreite benötigt. Für die Latenz muss eine Konfiguration gefunden werden, sodass diese möglichst niedrig ist. Eine Unterscheidung in einzelne Bereiche innerhalb der Systeme, in denen niedrige Latenz beziehungsweise hohe Bandbreite benötigt wird, ist nicht vorgesehen.
Zusammenfassend wird eine Konfiguration gesucht, die einem Teil des Cloud-Servers besondere Priorität verleiht. Auf dem anderen Teil des Servers soll dennoch normal gearbeitet werden können. Zusätzlich soll durch das Backup der Datenbanken und des Storages möglichst keine Verzögerung des anderen Verkehrs erzeugt werden.

Unabhängig des Anwendungsfalls werden die Auswirkungen von *QoS* nur in Überlastsituationen sichtbar. Gerade in performancekritischen Bereichen, in denen Infiniband meist eingesetzt wird, sollen Überlastsituationen generell verhindert werden. Dies wird unter anderem durch spezielle Netzwerk-Topologien wie der *Full-Fat-Tree-Topologie* erreicht. Durch diese wird ermöglicht, dass disjunkte Knotenpaare jeweils mit voller Bandbreite blockadefrei kommunizieren können. Das bedeutet, dass Überlastsituationen nur auftreten können, wenn durch mehrere Knoten auf einen einzelnen Knoten zugegriffen wird, wie es beispielsweise beim Zugriff auf eine Storage-Einheit der Fall ist. Des Weiteren können durch Defekte einzelner Kabel oder Ports Überlastfälle ausgelöst werden. Somit wird durch den Einsatz von *QoS* bei Verwendung von blockadefreien Topologien auch ein zusätzliches Sicherheitsinstrument etabliert. Dadurch wird in den Fällen, in denen durch die Topologie eine Blockade nicht verhindert werden kann, die Möglichkeit geschaffen, dass die Kapazitäten eines Netzwerkes beliebig auf einzelne Anwendungen verteilt werden können. Diese Zuteilung der Kapazität gilt auch im

Falle von Defekten, wodurch die Performance für die priorisierten Anwendungen stabiler gehalten werden kann. Außerdem können über *QoS* Ausnahmefälle, wie beispielsweise die unterschiedlich behandelten *VMs* im betrachteten Datacenter, definiert werden.
Ein weiterer Vorteil, den der Einsatz von *QoS* mit sich bringt, ist es, dass die tatsächlich benötigte Kapazität des Netzes besser abgeschätzt werden kann. Dies ist möglich, da durch *QoS* unabhängig von anderem Datenverkehr einzelnen Anwendungen beziehungsweise Protokollen gegenüber eine feste Größe, beispielsweise an Bandbreite, garantiert werden kann. Dadurch kann der benötigte Faktor für ein Overprovisioning, wodurch die Kapazität für den restlichen Verkehr bestimmt wird, besser festgelegt werden. Aufgrund der besseren Planbarkeit, können Kosten bei der Anschaffung eines Netzwerkes minimiert werden.

Für die Überprüfung der Realisierbarkeit der zu den Anwendungsbeispielen entwickelten Ideen wird in Kapitel 4.2 die vorhandene Testumgebung vorgestellt. Im Folgenden wird in Kapitel 4.3 die grundsätzliche Funktionalität von *QoS* auf Infiniband nachgewiesen. Darauf aufbauend werden in Kapitel 4.4 weitere Analysen bezüglich der Bandbreite, der Latenz, sowie der genaue Zuweisung der *SLs* durchgeführt. In Kapitel 4.5 werden die hier entwickelten Ideen mit einer entsprechend ausgearbeiteten Konfiguration umgesetzt.

4.2 Testcluster Juliette

Im Weiteren Verlauf dieser Arbeit werden Messungen zur Überprüfung und Bewertung von *QoS* auf Infiniband durchgeführt. Da die Konfiguration von *QoS*, sowie die Messungen an sich starke Auswirkungen auf den Betrieb eines Clusters haben, wird für diese Arbeit das Test-Cluster *Juliette* verwendet. *Juliette* ist die Kurzform von *Juelich Lustre Infiniband Environment To Test Enhancements*. Es steht für alle Messungen dieser Arbeit dediziert zur Verfügung.
Das Cluster ist mit acht Knoten des Typs *SGI-C1103-TY12* ausgerüstet. Sie besitzen jeweils einen *Intel Xeon* Prozessor des Typs *E5630* mit acht Kernen. Alle Kerne werden mit einer Taktfrequenz von 2,53 GHz betrieben. Jedem Knoten stehen 25 GB Arbeitsspeicher zur Verfügung. Zur Netzwerkanbindung besitzt jeder Knoten eine *Mellanox Connect-X2* Adapterkarte des Typs *MHQH19B-XTR*. Darauf sind ein 10 Gb/s Ethernet-Port für ein IP-Administrationsnetzwerk, sowie ein 40 Gb/s *Infiniband-QDR-Port* verbaut.
Für die Infiniband-Anbindung im Cluster werden acht Mellanox Switches des Typs *IS5022* verwendet. Sie bieten eine blockadefreie Weiterleitung mit bis zu 40 Gb/s pro Port. Zur besseren Vergleichbarkeit mit den eingesetzten Produktionssystemen ist eine *Full-Fat-Tree-Topologie* eingesetzt. Eine schematische Übersicht des Infiniband-Netzwerks des Clusters ist in Abbildung 4.4 dargestellt. Die farbliche Unterscheidung der Links dient lediglich zur besseren Übersicht.

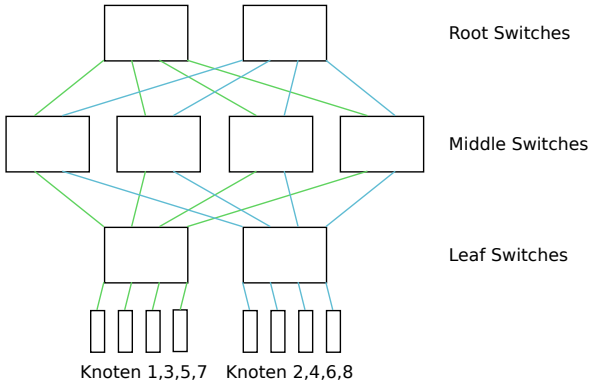

Abbildung 4.4: Schematischer Aufbau des Test-Clusters *Juliette*

Auf allen Knoten wird das Betriebssystem *CentOS 6.5* eingesetzt. Außerdem wird die *Mellanox-OFED-Version 2.4-1.0.0* verwendet. Auf den *HCAs* ist die aktuellste Firmwareversion *2.9.1000* installiert.
Alle Knoten wurden mittels des *Scaling Govenors* in den Performance-Modus versetzt, das heißt, dass immer die maximale Leistung der Prozessoren zur Verfügung steht. Dadurch wird sichergestellt, dass keine Schwankungen bei den Messungen durch verminderte Taktraten im Energiesparmodus entstehen. Des Weiteren wird auf allen Knoten eine *MTU* von 2048 Bytes festgelegt.

Mit den Standardkonfigurationen und der Verwendung von *RDMA* wird eine mittlere Bandbreite von 3,36 GB/s erreicht. Die Latenz variiert je nach Auswahl der Knoten zwischen 10,9 μs und 11,3 μs. Werden zwei Knoten mit gerader beziehungsweise ungerader Knotennummer ausgewählt, so sind sie direkt über einen *Leaf Switch* verbunden. Daher werden nur zwei Hops benötigt. Bei Auswahl eines Knoten mit gerader und einem Knoten mit ungerader Knotennummer werden jeweils vier Hops benötigt. Im weiteren Verlauf werden alle Messungen auf Knoten mit unterschiedlichen *Leaf Switches* durchgeführt, sodass die Auswirkungen von *QoS* deutlicher werden. Deshalb werden die Werte 3,36 GB/s und 11,3 μs als Referenzen verwendet.

4.3 Proof of Concept

Ziel dieses Kapitels ist es, die grundsätzliche Funktionalität von *QoS* auf Infiniband nachzuweisen. Dazu wird zunächst die Zuweisung von *SLs* auf Basis der Kommandozeile, sowie die Umsetzung einer *SL* auf eine *VL* überprüft. Im weiteren Verlauf werden rudimentäre Konfigurationsmöglichkeiten bezüglich des *SL to VL Mappings* anhand einer Bandbreitenmessung betrachtet. Schließlich wird die automatisierte Zuweisung der *SLs* aufgrund einer Konfigurationsdatei in der *Simple-* und *Advanced-Syntax* getestet.
Dazu werden vor allem die Werkzeuge *ibdump* und *qperf* verwendet. Durch *ibdump* kann auf eine ähnliche Funktionalität, wie sie durch das bekannte *tcpdump* geliefert wird, zurückgegriffen werden. Es können daher alle über einen Link versendeten Pakete protokolliert und mitgeschnitten werden. Dabei stehen besonders die Header-Daten bezüglich der genutzten *VL* und des eingetragenen *SL* im Vordergrund. Auf Basis der von *ibdump* erzeugten Daten, kann eine graphische Auswertung mit dem Werkzeug *wireshark* erfolgen. Dadurch wird eine bessere Übersicht, sowie ein einfacher Zugriff auf die versendeten Header-Daten ermöglicht.
Das Werkzeug *qperf* bietet, im Gegensatz zu dem bekannteren *iperf*, eine Unterstützung für die Nutzung von *RDMA*. Ansonsten ist das Verhalten der beiden Werkzeuge sehr ähnlich.

Für den ersten Test bezüglich der korrekten Zuweisung des *SL* und der Nutzung der entsprechenden *VL*, wird eine rudimentäre Konfiguration des *SM* vorgenommen. Sie ist in Abbildung 4.5 dargestellt. Es werden maximal fünf *VLs* zugelassen

```
qos_max_vls 5
qos_high_limit 100
qos_vlarb_high 0:100
qos_vlarb_low 1:64,2:64,3:64,4:64
qos_sl2vl 0,1,2,3,4,15,15,15,15,15,15,15,15,15,15
```

Abbildung 4.5: Einfache Konfiguration in *opensm.conf*

und ein *High-Limit* von 100 gesetzt. Von den *VLs* wird nur *VL 0* mit einem Gewicht von 100 hoch priorisiert. Alle anderen *VLs* werden mit einem jeweiligen Gewicht von 64 niedrig priorisiert. Die *SL 0-4* werden auf die entsprechenden *VL 0-4* abgebildet. Allen anderen *SLs* wird *VL 15* zugeordnet, wodurch sie blockiert werden.
Die Konfiguration des *SL to VL Mappings*, sowie die der *VL Arbitration* kann auf jeder Komponente mittels des Kommandos *smpquery sl2vl* beziehungsweise *smpquery vlarb* abgefragt werden. Das aus der gerade vorgestellten Konfiguration resultierende Ergebnis des *SL to VL Mappings* ist in Abbildung 4.6 für einen

Knoten dargestellt. Im Beispiel wurde der Knoten mit der *LID 11* an *Port 0* abgefragt. Für den Port ist das Mapping jedes *SL* separat dargestellt. Das Ergebnis

```
# SL2VL table: Lid 11
#                  SL: | 0| 1| 2| 3| 4| 5| 6| 7| 8| 9|10|11|12|13|14|15|
ports: in  0, out  0: | 0| 1| 2| 3| 4|15|15|15|15|15|15|15|15|15|15|15|
```

Abbildung 4.6: Abfrage des *SL to VL Mappings* eines Knotens mittels *smpquery*

entspricht der vorgenommenen Konfiguration im *SM*. Bei einem Switch würden, entsprechend der Anzahl an Ports, mehr Zeilen in der Tabelle angezeigt werden. Die abgefragte *VL Arbitration* des selben Knotens ist in Abbildung 4.7 dargestellt. Es werden separat die niedrig und die hoch priorisierte Liste angezeigt. Für jede Liste wird tabellarisch jeder *VL* das entsprechende Gewicht zugeordnet. Auch hier wurde die Konfiguration korrekt umgesetzt.

```
# VLArbitration tables: Lid 11 port 0 LowCap 8 HighCap 8

# Low priority VL Arbitration Table:
VL     : |0x1 |0x2 |0x3 |0x4 |0x0 |0x0 |0x0 |0x0 |
WEIGHT: |0x40|0x40|0x40|0x40|0x0 |0x0 |0x0 |0x0 |

# High priority VL Arbitration Table:
VL     : |0x0 |0x0 |0x0 |0x0 |0x0 |0x0 |0x0 |0x0 |
WEIGHT: |0x64|0x0 |0x0 |0x0 |0x0 |0x0 |0x0 |0x0 |
```

Abbildung 4.7: Abfrage des *VL Arbitration* eines Knotens mittels *smpquery*

Die Konfiguration der *Policies* erfolgt mittels der *Simple-Syntax* und definiert nur einen Default, durch den allen Datenströmen das *SL 0* zugewiesen wird. Die Konfiguration ist in Abbildung 4.8 dargestellt. Für die Erzeugung eines Daten-

```
qos-ulps
   default : 0
end-qos-ulps
```

Abbildung 4.8: Minimale Konfiguration der *Policies*

stroms wird eine Bandbreitenmessung mit dem Werkzeug *qperf* vorgenommen. Dazu wird clientseitig (auf Knoten 7) das Werkzeug, wie in Abbildung 4.9 dargestellt, verwendet. Das bedeutet, dass eine Bandbreitenmessung mittels *RDMA* mit dem Infiniband-Interface von *Knoten 6* vollzogen wird. Dabei werden alle Pakete mit dem *SL 1* markiert.
Parallel zur Messung wird serverseitig der Verkehr mittels *ibdump* aufgezeichnet. Beispielhaft wird in Abbildung 4.10 ein Paket aus der Messung, wie in *wireshark* angezeigt, dargestellt. Das erwartete *SL 1* und auch die entsprechend kon-

```
qperf n006-ib -sl 1 rc_rdma_write_bw
```

Abbildung 4.9: Clientseitiger Aufruf von *qperf*

```
Frame 65: 2074 bytes on wire (16592 bits), 2074 bytes captured (16592 bits) on interface 0
Extensible Record Format
InfiniBand
  Local Route Header
    0001 .... = Virtual Lane: 0x01
    .... 0000 = Link Version: 0
    0001 .... = Service Level: 1
    .... 00.. = Reserved (2 bits): 0
    ...... 10 = Link Next Header: 0x02
    Destination Local ID: 11
    0000 0... .... .... .... = Reserved (5 bits): 0
    .... .010 0000 0110 = Packet Length: 518
    Source Local ID: 15
  Base Transport Header
  IBA Payload - appears to be EtherType encapsulated
  Invariant CRC: 0x91b28151
  Variant CRC: 0x0b8e
MDS Header(Unknown(0)/Unknown(11))
Fibre Channel
Data (2010 bytes)
```

Abbildung 4.10: Darstellung eines Paketes in *wireshark*

figurierte *VL 1* werden verwendet. Sie sind mittels eines *roten* Rahmens in der Abbildung hervorgehoben.
Wird beim Aufruf von *qperf* ein über *VL 15* blockiertes *SL*, beispielsweise *SL 5* verwendet, kann keine Verbindung zum anderen Knoten aufgebaut werden. Die Blockierung einer *SL* über *VL 15* wird dadurch bestätigt.

Durch den zweiten Test soll der Einfluss der *VL Arbitration* bei paralleler Verwendung von mehreren *SLs* überprüft werden. Dies ist ein zentraler Punkt bei der Betrachtung von *QoS*, da dadurch die differenzierte Behandlung einzelner Anwendungen ermöglicht wird. Dazu wird eine etwas andere Konfiguration des *SM* als im vorherigen Test benötigt. Die Änderungen sind in Abbildung 4.11

```
qos_high_limit 255
qos_vlarb_high 1:64,2:32
qos_vlarb_low 0:1,1:0,4:0
```

Abbildung 4.11: Unterschiedliche Gewichtung der *VLs*

dargestellt. Es wird nun ein *High-Limit* von 255 verwendet. Dadurch wird eine Behinderung der Messungen durch die niedrig priorisierten Liste ausgeschlossen. Die *VLs 1* und *2* werden mit den Gewichten 64 und 32 hoch priorisiert. Alle anderen *VLs* werden niedrig priorisiert. Bis auf *VL 0* erhalten alle ein Gewicht von 0. Sie werden daher nicht weitergeleitet. *VL 0* wird ein Gewicht von eins zugeordnet, damit etwaiger anfallender Default-Verkehr bei freier Kapazität ge-

sendet werden kann. Es wird demnach erwartet, dass bei gleichzeitiger Nutzung eines Links $\frac{2}{3}$ der Bandbreite auf *SL 1* und $\frac{1}{3}$ auf *SL 2* entfallen.
Zur Überprüfung wird wiederum eine Bandbreitenmessung mittels *qperf* verwendet. Wie in Abbildung 4.12 dargestellt, werden dazu von einem Knoten aus zwei Prozesse mit je einer Messung und unterschiedlichem *SL* gestartet. Dadurch wird

```
qperf n007-ib -sl 1 rc_rdma_write_bw &
 qperf n008-ib -sl 2 rc_rdma_write_bw
```

Abbildung 4.12: Parallele Bandbreitenmessung mit zweier unterschiedlicher *SLs*

sichergestellt, dass auf dem Link zwischen Knoten 6 und dem ersten Switch eine Überlastsituation herrscht. Somit können die durch *QoS* verursachten Effekte beobachtet werden.
Das in Abbildung 4.13 dargestellte Ergebnis, liefert bis auf eine Rundung der

```
[n007-ib] rc_rdma_write_bw: bw  =  2.33 GB/sec
[n008-ib] rc_rdma_write_bw: bw  =  1.17 GB/sec
```

Abbildung 4.13: Ergebnis der parallelen Messung mit zwei unterschiedlichen *SLs*

Werte das erwartete Verhältnis von zwei zu eins zwischen *SL 1* und *SL 2*. Das Ergebnis wurde durch mehrere unabhängige Messungen bestätigt. Außerdem wird durch die korrekte Umsetzung der Gewichtung zusätzlich die richtige Markierung der Pakete nochmals indirekt nachgewiesen.

Für einen möglichen späteren Einsatz von *QoS* spielt die automatische Zuweisung der *SLs* auf die einzelnen Anwendungen eine zentrale Rolle, da darüber die *Isolation von Scheduling und Policing* umgesetzt wird. Durch den nachfolgenden Test wird die grundsätzliche Funktionalität der *Policies* belegt.
Dafür wird zunächst eine schlichte Konfiguration einer *Policy* in der *Simple-Syntax* vorgenommen. Sie ist in Abbildung 4.14 dargestellt. Darin wird für jeg-

```
qos-ulps
   ipoib : 1
   default : 0
end-qos-ulps
```

Abbildung 4.14: Konfiguration eines *SL* für *IPoIB*

liche Verwendung von *IPoIB* das *SL 1* festgelegt. Ansonsten wird immer der Default (*SL 0*) verwendet.

Alle anderen Konfigurationen werden aus dem vorherigen Test übernommen.
Zur Kontrolle der korrekten Zuweisung des *SL 1* für eine Anwendung die *IPoIB* nutzt, wird das Werkzeug *iperf* verwendet. Wie in Abbildung 4.15 dargestellt, wird auf Basis von *TCP* eine Bandbreitenmessung zu Knoten 6 vorgenommen. Parallel dazu wird mittels *ibdump* ein Mitschnitt aller versendeter Pakete erstellt. Zur Überprüfung der gesetzten *SLs*, beziehungsweise der genutzten *VLs*,

```
iperf -c n006-ib
```

Abbildung 4.15: Verwendung von *iperf* zur Bandbreitenmessung

wird wiederum *wireshark* verwendet. In Abbildung 4.16 ist ein Paket des erzeugten Datenstroms dargestellt. Darin wird das verwendete *SL*, sowie die genutzte

```
Frame 38: 2082 bytes on wire (16656 bits), 2082 bytes captured (16656 bits) on interface 0
Extensible Record Format
InfiniBand
    Local Route Header
        0001 .... = Virtual Lane: 0x01
        .... 0000 = Link Version: 0
        0001 .... = Service Level: 1
        .... 00.. = Reserved (2 bits): 0
        .... ..10 = Link Next Header: 0x02
        Destination Local ID: 13
        0000 0... .... .... = Reserved (5 bits): 0
        .... .010 0000 1000 = Packet Length: 520
        Source Local ID: 15
    Base Transport Header
    DETH - Datagram Extended Transport Header
    IBA Payload - appears to be EtherType encapsulated
    Invariant CRC: 0xf42c535b
    Variant CRC: 0xc1d3
Internet Protocol Version 4, Src: 192.168.120.18 (192.168.120.18), Dst: 192.168.120.17 (192.168.120.17)
Transmission Control Protocol, Src Port: 56030 (56030), Dst Port: commplex-link (5001), Seq: 25, Ack: 1, Len: 2004
Data (2004 bytes)
```

Abbildung 4.16: Darstellung eines *IPoIB-TCP-Paketes* in *wireshark*

VL mit einem *roten* Rahmen markiert. Zusätzlich ist die Verwendung von *IP* hervorgehoben.
Neben allen Datenpaketen werden auch die Bestätigungen, die vom Server an den Client geschickt werden, mit *SL 1* markiert. Dadurch wird die korrekte Umsetzung der getroffenen Regelung bestätigt.

Zur weiteren Überprüfung wird die *Advanced-Syntax* für die *Policies* verwendet. Durch die in Abbildung 4.17 dargestellte Konfiguration wird der von einem Knoten ausgehende Verkehr separiert. Dazu wird eine Port-Gruppe *Node7* erstellt, die nur Knoten 7 enthält. Neben dem Default-Level wird ein zweites *QoS-Level* namens *Level2* definiert. Darin wird das *SL 2* zugewiesen. In der einzigen Regel wird schließlich dem Datenverkehr mit der Quelle *Node7* das *Level2* zugeordnet. Die Konfiguration wird wiederum mittels einer Bandbreitenmessung durch *iperf* und der entsprechenden Aufzeichnung der Pakete zwischen den Knoten 6 und

```
port-groups
  port-group
    name: Node7
    port-guid: 0x002590ffff2fb1f5
  end-port-group
end-port-groups

qos-levels
  qos-level
    name: DEFAULT
    sl: 1
  end-qos-level
  qos-level
    name: Level2
    sl: 2
  end-qos-level
end-qos-levels

qos-match-rules
  qos-match-rule
    source: Node7
    qos-level-name: Level2
  end-qos-match-rule
end-qos-match-rules
```

Abbildung 4.17: Separierung eines Quellknotens in der *Advanced-Syntax*

7 überprüft. Durch die Auswertung in *wireshark* wird das erwartete Verhalten nachgewiesen. Alle Pakete die von Knoten 7 gesendet werden, werden mit dem *SL 2* markiert und über *VL 2* versendet. Die Pakete, die von Knoten 6 an Knoten 7 geschickt werden, nutzen das Default-Level und werden dementsprechend mit dem *SL 1* markiert. Zur Verdeutlichung ist in Abbildung 4.18 ein Paket von Knoten 7 an Knoten 6 und in Abbildung 4.19 die dazugehörige Bestätigung von Knoten 6 an Knoten 7 dargestellt. Zur besseren Anschaulichkeit sind das jeweils verwendete *SL* beziehungsweise *VL* mit einem roten Rahmen markiert. Zusätzlich sind die IP-Adressen des Senders beziehungsweise des Empfängers hervorgehoben. Es ist zu erkennen, dass sobald Knoten 7 (IP: 192.168.120.18) sendet, das *SL 2* und somit auch die *VL 2* verwendet wird. Sendet Knoten 6 (IP: 192.168.120.17), wird *SL 1* und *VL 1* genutzt. Dies entspricht genau dem nach der Konfiguration zu erwartenden Verhalten.

Durch die in diesem Kapitel ausgeführten Tests konnte die grundsätzliche Funktionalität von *QoS* auf Infiniband nachgewiesen werden.
Im folgenden Kapitel 4.4 werden weitere Analysen zum Verhalten von *QoS* auf Infiniband bezüglich der Bandbreite, Latenz und der Zuweisung der *SL* durchgeführt.

```
Frame 151: 2082 bytes on wire (16656 bits), 2082 bytes captured (16656 bits) on interface 0
Extensible Record Format
InfiniBand
    Local Route Header
        0001 .... = Virtual Lane: 0x02
        .... 0000 = Link Version: 0
        0001 .... = Service Level: 2
        .... 00.. = Reserved (2 bits): 0
        .... ..10 = Link Next Header: 0x02
        Destination Local ID: 13
        0000 0... .... .... = Reserved (5 bits): 0
        .... .010 0000 1000 = Packet Length: 520
        Source Local ID: 15
    Base Transport Header
    DETH - Datagram Extended Transport Header
    IBA Payload - appears to be EtherType encapsulated
    Invariant CRC: 0x6d8d9e45
    Variant CRC: 0xc548
Internet Protocol Version 4, Src: 192.168.120.18 (192.168.120.18), Dst: 192.168.120.17 (192.168.120.17)
Transmission Control Protocol, Src Port: 56284 (56284), Dst Port: commplex-link (5001), Seq: 25259, Ack: 1, Len: 2004
Data (2004 bytes)
```

Abbildung 4.18: Datenpaket von Knoten 7 an Knoten 6

```
Frame 152: 78 bytes on wire (624 bits), 78 bytes captured (624 bits) on interface 0
Extensible Record Format
InfiniBand
    Local Route Header
        0001 .... = Virtual Lane: 0x01
        .... 0000 = Link Version: 0
        0001 .... = Service Level: 1
        .... 00.. = Reserved (2 bits): 0
        .... ..10 = Link Next Header: 0x02
        Destination Local ID: 15
        0000 0... .... .... = Reserved (5 bits): 0
        .... .000 0001 0011 = Packet Length: 19
        Source Local ID: 13
    Base Transport Header
    DETH - Datagram Extended Transport Header
    IBA Payload - appears to be EtherType encapsulated
    Invariant CRC: 0x2a73b3d5
    Variant CRC: 0x57a9
Internet Protocol Version 4, Src: 192.168.120.17 (192.168.120.17), Dst: 192.168.120.18 (192.168.120.18)
Transmission Control Protocol, Src Port: commplex-link (5001), Dst Port: 56284 (56284), Seq: 1, Ack: 232489, Len: 0
```

Abbildung 4.19: Bestätigung von Knoten 6 an Knoten 7

4.4 Weitere Analysen

4.4.1 Bandbreite

In diesem Abschnitt werden weitere Analysen zur Verteilung der Bandbreite auf unterschiedliche *SLs* vorgenommen. Ziel ist es dabei möglichst viele Informationen über die Auswirkungen der einzelnen Parameter zu gewinnen. Alle Messungen in diesem Abschnitt werden für mindestens zehn Sekunden ausgeführt und jeweils mehrmals separat wiederholt. Die dargestellten Messwerte sind die daraus resultierenden Durchschnittswerte.

Zunächst wird die Auswirkung durch die Verwendung von mehreren *VLs* überprüft. Dazu werden zwischen zwei Knoten mehrere Bandbreitenmessungen mittels *qperf* vorgenommen. Im Verlauf wird die Anzahl paralleler Messungen und somit auch die Anzahl an parallelen Datenströmen erhöht. Es werden die Ergebnisse der jeweils parallel ausgeführten Messungen summiert, um einen Vergleich zur Bandbreite bei nur einem Datenstrom treffen zu können. Zur Messung wird *qperf* verwendet, um jeglichen Overhead durch *ULPs* ausschließen zu können.
Die Konfiguration des *SM* wird, wie in Abbildung 4.20 dargestellt, vorgenommen. Es wird mit maximal fünf *VLs* gearbeitet. Das *qos_high_limit* wird auf 255

```
qos_max_vls 5
qos_high_limit 255
qos_vlarb_high 1:128,2:128,3:128,4:128
qos_vlarb_low 0:1
qos_sl2vl 0,1,2,3,4,15,15,15,15,15,15,15,15,15,15,15
```

Abbildung 4.20: Konfiguration des *SM* für die erste Bandbreitenmessung

gesetzt, wodurch dem hoch priorisierten Verkehr absoluter Vorrang gewährt wird. Von den fünf *VLs* werden *VL 1-4* hoch priorisiert und mit jeweils 128 gleich gewichtet. *VL 0* wird mit einem Gewicht von eins niedrig priorisiert. Die *SLs 1-4* werden entsprechend den *VLs 1-4* zugewiesen. Alle anderen *SLs* werden über *VL 15* blockiert.
Zusätzlich wird nur eine *Default-Policy* definiert, die jeglichem Verkehr das *SL 0* zuweist. Durch die Konfiguration wird sichergestellt, dass für die Messung vier separate, gleich gewichtete *VLs* zur Verfügung stehen, die von keinem anderen Verkehr behindert werden können.
In Abbildung 4.21 sind zwei Messreihen dargestellt. Die *rote* Kurve stellt eine Referenzmessung ohne die Verwendung von unterschiedlichen *VLs* dar. Im Vergleich dazu steht die *grüne* Kurve, in der für jede einzelne Messung eine andere *VL* verwendet wurde. Generell zeigt sich, dass bei der vorgenommenen Konfiguration die Bandbreite auch bei nur einer parallelen Messung 3,38 GB/s beträgt und nicht, wie zuvor ohne *QoS*, 3,36 GB/s. Aus der Referenzmessung ist zu

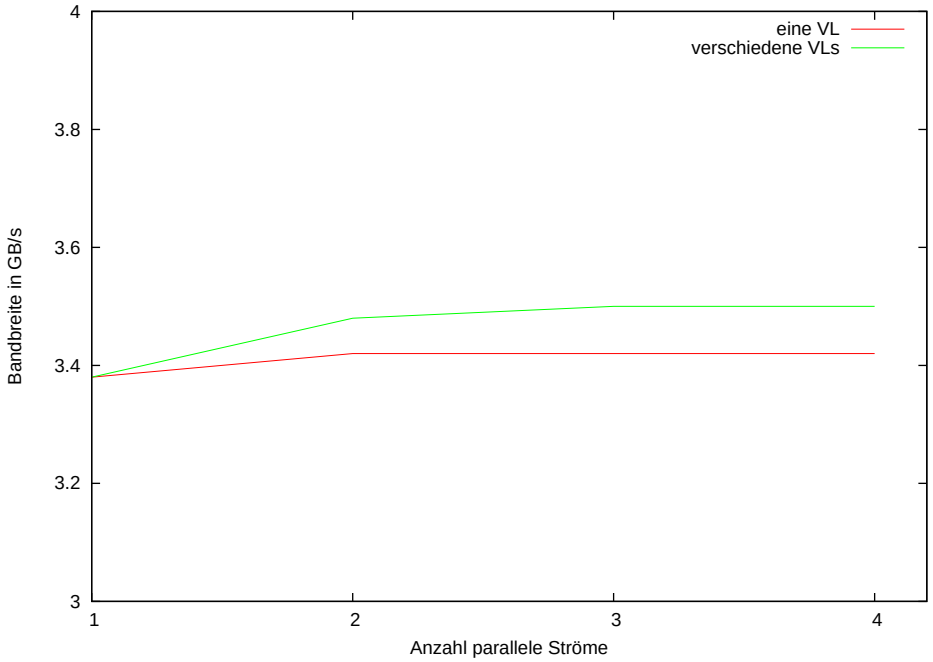

Abbildung 4.21: Summierte Bandbreite bei mehreren parallelen Messungen

erkennen, dass bei paralleler Verwendung eines Links durch mindestens zwei Datenströme, ein weiterer Zuwachs der summierten Bandbreite von 3,38 GB/s auf 3,42 GB/s vorliegt. Werden zusätzlich unterschiedliche *VLs* verwendet, wird eine noch höhere summierte Bandbreite erzielt. Ab drei parallelen Datenströmen mit unterschiedlichen *VLs* werden bis zu 3,50 GB/s gemessen. Das entspricht einem Zuwachs im Vergleich zu einem einzelnen Datenstrom ohne Verwendung von *QoS* um 4,17 %.

Zur Überprüfung des Einflusses unterschiedlicher Gewichte pro *VL* auf die gewonnenen Erkenntnisse wird eine weitere Messung durchgeführt. Dazu werden die Gewichte der *VL 1-4*, wie in Abbildung 4.22 dargestellt, geändert. Dadurch

```
qos_vlarb_high 1:128,2:64,3:32,4:16
```

Abbildung 4.22: Konfiguration des *SM* für die zweite Bandbreitenmessung

wird eine Staffelung der Gewichte der *VLs* im jeweiligen Verhältnis von zwei zu eins vorgenommen.

In der Folge wird die erste Messung mit den neuen Gewichten wiederholt. In Abbildung 4.23 ist das Ergebnis dargestellt. Dazu wird zur Veranschaulichung die *rote* Kurve, sowie die Messpunkte der *grünen* Kurve aus Abbildung 4.21 erneut eingezeichnet. Für die neuen Messwerte wird die aufsummierte Bandbreite

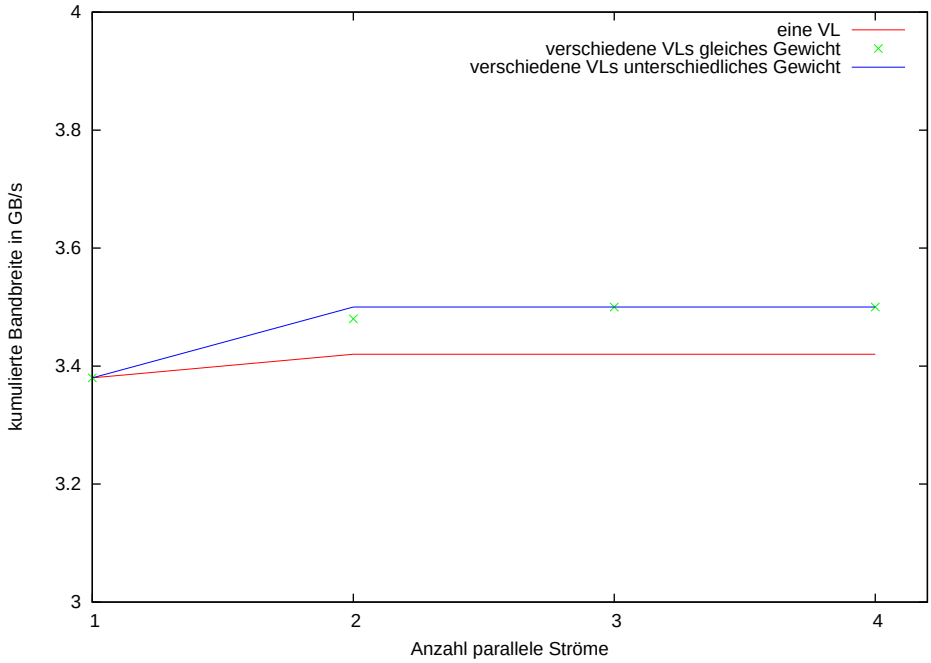

Abbildung 4.23: Summierte Bandbreite bei mehreren parallelen Messungen mit unterschiedlichem Gewicht

als *blaue* Kurve dargestellt. Dadurch wird deutlich, dass bis auf eine kleine Abweichung bei zwei *VLs* verschiedene Gewichte keinen Einfluss auf die summierte Bandbreite haben. Es gilt also weiterhin, dass bei paralleler Verwendung von mehreren *VLs* die maximale Bandbreite höher ist, als bei einer *VL*.

Im Folgenden wird der Einfluss unterschiedlicher Nachrichtengrößen auf die genutzte Bandbreite betrachtet. Dazu wird die Konfiguration aus dem vorhergehenden Beispiel beibehalten. Es können demnach vier verschiedene *VLs* verwendet werden, die in ihrem Gewicht im jeweiligen Verhältnis von zwei zu eins abgestuft sind.
Für die Messungen werden drei der vier *VLs* mittels einer Bandbreitenmessung durch *qperf* voll belastet. Auf der jeweiligen unbelasteten *VL* wird eine Bandbreitenmessung mit *ib_write_bw*, einem Diagnosewerkzeug von *Mellanox*, durchge-

führt. Dabei werden sukzessiv die Nachrichtengrößen von 2 Bytes auf 2^{23} Bytes erhöht.
In Abbildung 4.24 sind die Ergebnisse dargestellt. Dazu wird die gemessene Bandbreite der einzelnen *VLs* gegen die Nachrichtengröße aufgetragen. Zur besse-

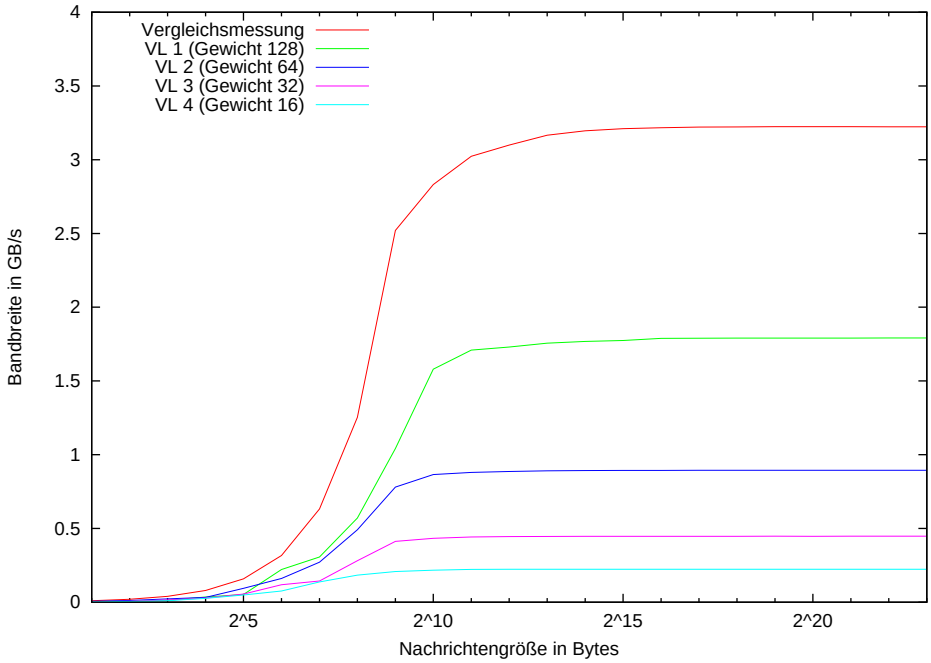

Abbildung 4.24: Verlauf der Bandbreite bei verschiedenen Nachrichtengrößen

ren Vergleichbarkeit wurde zusätzlich eine Messung auf einem unbelasteten Link durch *ib_write_bw* erstellt. Diese ist *rot* gekennzeichnet.
Im Allgemeinen ist zu erkennen, dass bei kleinen Nachrichtengrößen nicht die volle Bandbreite ausgenutzt werden kann. Daher spielt in diesem Bereich auch die Gewichtung der *VLs* keine große Rolle. Erst ab einer gewissen Größe, die in Abhängigkeit zur maximalen Bandbreite der *VL* steht, wird ein größerer Unterschied zwischen den *VLs* deutlich. Mit zunehmender Größe wird eine Annäherung der genutzten Bandbreite der einzelnen *VLs* an das jeweilig konfigurierte Gewicht deutlich. Da der Link unter Überlast steht, wird die dem Gewicht entsprechende Bandbreite nicht überschritten.

Als weiterer Parameter hat das *High-Limit* einen großen Einfluss auf die Zuweisung der Bandbreite, da darüber das Verhältnis zwischen hoch und niedrig priorisierten Datenströmen festgelegt werden kann. Dafür wird die Konfigurati-

on der *VL-Arbitration*, wie in Abbildung 4.25 dargestellt, geändert. Es werden wiederum maximal fünf verschiedenen *VLs* zugelassen. Drei *VLs* werden mit einem Gewicht von 128 hoch priorisiert und eine ebenfalls mit einem Gewicht von 128 niedrig priorisiert. Zusätzlich ist *VL 0* für den Default-Verkehr mit einem

```
qos_max_vls 5
qos_vlarb_high 2:128,3:128,4:128
qos_vlarb_low 1:128, 0:1
qos_sl2vl 0,1,2,3,4,15,15,15,15,15,15,15,15,15,15,15
```

Abbildung 4.25: Konfiguration des *SM* für die vierte Bandbreitenmessung

Gewicht von eins niedrig priorisiert. Dadurch wird der Einfluss von anderem Verkehr auf die Messung nahezu ausgeschlossen. Das *SL to VL Mapping* wird wie in den vorherigen Messungen vorgenommen.
Für die Messung wird eine Bandbreitenmessung mittels *qperf* auf der niedrig priorisierten *VL* ausgeführt. Im weiteren Verlauf werden parallel dazu immer mehr hoch priorisierte *VLs*, ebenfalls durch Bandbreitenmessungen, belastet. Die Messreihen werden mit unterschiedlichen Werten für das *High-Limit* wiederholt. Das Ergebnis der Messungen ist in Abbildung 4.26 dargestellt. Darin wird für die unterschiedliche Werte des *High-Limits* die gemessene Bandbreite der niedrig priorisierten *VL* gegen die Anzahl der konkurrierenden hoch priorisierten Datenströme aufgetragen.
Das Ergebnis entspricht nicht den nach dem Standard erwarteten Werten. So müsste bei einem Limit von null jedes zweite Paket aus der niedrig priorisierten Liste versendet werden. Da nur über eine niedrig priorisierte *VL* gesendet wird, müsste unabhängig von der Anzahl an konkurrierenden hoch priorisierten Datenströmen, die Hälfte der zur Verfügung stehenden Bandbreite auf dieser *VL* nachgewiesen werden können. Als Basis für die Berechnungen wird bei mehreren Datenströmen von einer maximalen summierten Bandbreite von 3,5 GB/s ausgegangen. Im Beispiel müsste daher die gemessene Bandbreite der niedrig priorisierten *VL* von oben gegen eine Schranke von 1,75 GB/s laufen. Stattdessen wird bei zwei konkurrierenden Strömen nur ein Drittel und bei drei Strömen nur ein Viertel der Bandbreite erreicht.
Durch die Messung mit einem Limit von zehn wird eine weiter Auffälligkeit beobachtet. Während bei einem Limit von null und einem konkurrierenden Datenstrom noch genau die Hälfte der Bandbreite erreicht wird, wird bei einem Limit von zehn in etwa ein Elftel erwartet, da jedes elfte Paket aus der niedrig priorisierten *VL* gesendet wird. Dies entspricht circa 9,09 % (ca. 318 MB/s) der Bandbreite. Die Messung ergibt mit einem Wert von 491 MB/s jedoch 14,03 % der Bandbreite. Werden weitere hoch priorisierte Ströme hinzugefügt, wird festgestellt, dass die Schranke von circa 318 MB/s ebenfalls nicht eingehalten wird. Ähnliches Verhalten wird auch bei einem Limit von 125 beobachtet.

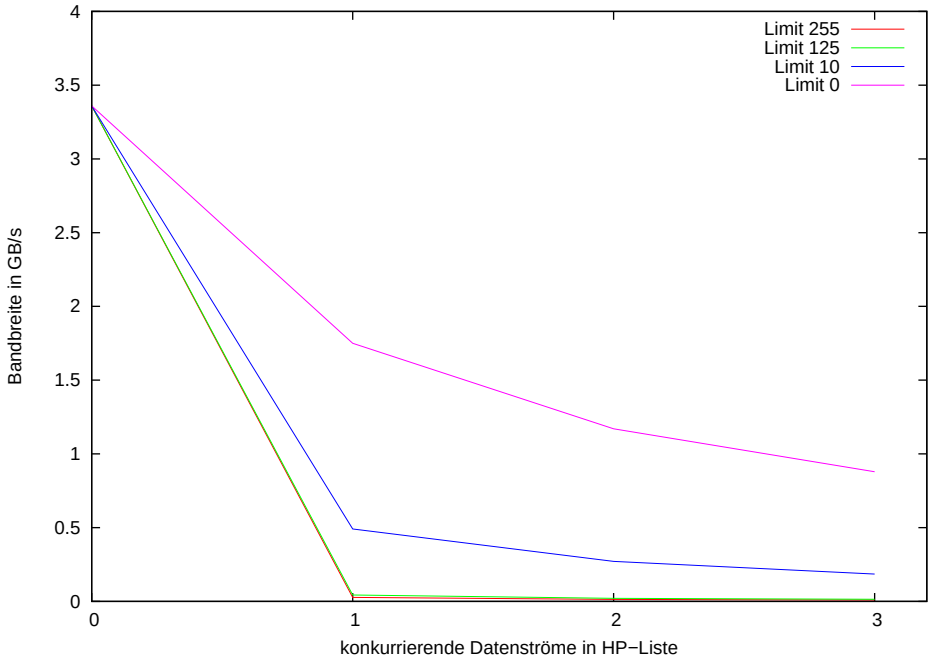

Abbildung 4.26: Einfluss des Parameters *qos_high_limit*

Durch ein Limit von 255 wird laut Standard definiert, dass nur niedrig priorisierte Pakete verschickt werden, wenn keine hoch priorisierten auf das Versenden warten. Daher wird erwartet, dass bei mehreren konkurrierenden Strömen der niedrig priorisierte Verkehr komplett abgeschnitten wird. Auch dies konnte durch die Messung nicht bestätigt werden. Selbst bei drei hoch priorisierten Strömen wurde noch eine Bandbreite von 9,24 MB/s gemessen. Dies entspricht immerhin 0,264 % der gesamten Bandbreite.

Zusammenfassend wurde festgestellt, dass der Parameter *High-Limit* nicht wie im Standard beschrieben funktioniert. Darüber definierte Schranken für den niedrig priorisierten Verkehr werden nicht eingehalten. Eine grobe Zuweisung der Bandbreite zwischen hoch und niedrig priorisiertem Verkehr ist dennoch möglich. Allerdings ist die tatsächlich zugewiesene Bandbreite abhängig von der Anzahl der genutzten hoch priorisierten *VLs*. Das bedeutet, dass eine gleichzeitige Zuweisung der Bandbreite für ein hoch priorisierte und eine niedrig priorisierte *VL* nicht exakt vorgenommen werden kann. Dies gilt vor allem, wenn sowohl über die *VLs* der hoch als auch der niedrig priorisierten Liste viele Daten verschickt werden und somit viel Bandbreite benötigt wird. Das heißt, dass für Anwendungen

mit hohem Datenaufkommen nur eine der beiden Listen verwendet werden sollte. Dies entspricht einer nicht unerheblichen Einschränkung der Funktionalität von *QoS* auf Infiniband.

4.4.2 Latenz

Im Folgenden wird die Auswirkung der Konfigurationsparameter auf die Latenz untersucht. Dafür wird mit unterschiedlichen Konfigurationen jeweils eine *RDMA-Latenzmessung* mittels *qperf* vorgenommen. Eine volle Auslastung eines Links mit einer Latenzmessung ist nicht möglich, da nur Pakete mit keinem *Data-Payload* verschickt werden. Daher wird zur parallelen Belastung weiterhin eine *RDMA-Bandbreitenmessung* durch *qperf* verwendet. Dadurch wird der Link mit Paketen mit maximalem *Data-Payload* voll belastet. Soweit nicht anderes erwähnt wird jede Messung beziehungsweise Belastung auf einer separaten *VL* durchgeführt. Zur Vermeidung von Messfehlern werden alle Latenzmessungen über zehn Sekunden durchgeführt und mindestens drei Mal wiederholt. Die dargestellten Werte sind die daraus resultierenden Durchschnittswerte. Die Messungen werden zwischen Knoten 6 und Knoten 7 durchgeführt, daher werden vier Hops vom einem zum anderen Knoten benötigt.

Für die erste Messreihe wird der *SM* wie in Abbildung 4.27 dargestellt konfiguriert. Es werden maximal fünf *VLs* verwendet. Vier der fünf *VLs* werden jeweils

```
qos_max_vls 5
qos_high_limit 255
qos_vlarb_high 1:128,2:128,3:128,4:128
qos_vlarb_low 0:1
qos_sl2vl 0,1,2,3,4,15,15,15,15,15,15,15,15,15,15,15
```

Abbildung 4.27: Konfiguration des *SM* für die erste Latenzmessung

mit einem Gewicht von 128 hoch priorisiert. *VL 0* wird mit einem Gewicht von 1 niedrig priorisiert, um den Default-Verkehr aufzufangen. Die *SL 0-4* werden den entsprechenden *VL 0-4* zugeordnet. Alle anderen *SLs* werden über *VL 15* blockiert.
In Abbildung 4.28 sind die ersten Ergebnisse der Latenzmessungen auf *VL 1* dargestellt. Die *rote* Kurve stellt die Messung mit der oben beschriebenen Konfiguration dar. Für die anderen beiden Messungen wurde die Gewichtung der *VL*, auf der die Latenzmessung durchgeführt wird, verändert. Der *grünen* Kurve liegt ein deutlich reduziertes Gewicht von acht zu Grunde. Ein signifikanter Unterschied zur vorhergehenden Messung wird nicht festgestellt.
Für die Daten der *blauen* Kurve wird die hoch priorisierte Liste erneut modifiziert. Die Änderung ist in Abbildung 4.29 dargestellt. Aufgrund dessen wird eine niedrigere und konstantere Latenz erwartet. Dies liegt daran, dass durch die

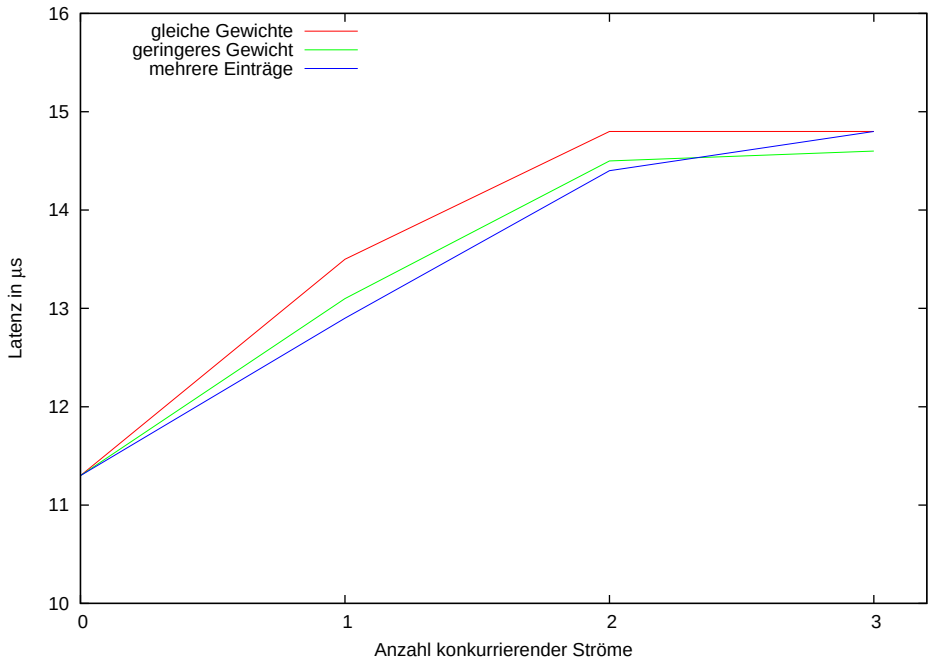

Abbildung 4.28: Latenz bei unterschiedlichen Gewichten

mehrfache Aufführung der *VL 1* in der hoch priorisierten Liste, die Zeit zwischen dem Versenden von Paketen aus *VL 1* geringer ist. Zur Vermeidung von

```
qos_vlarb_high 1:8,2:128,1:8,3:128,1:8,4:128
```

Abbildung 4.29: Änderung der Konfiguration des *SM* für die dritte Latenzmessung

Seiteneffekten wurde die Konfiguration je nach Anzahl der verwendeten *VLs* in der Messung angepasst, sodass pro genutzter anderer *VL* auch nur ein Eintrag für *VL 1* vorgenommen wird. Allerdings ist auch mit dieser Konfiguration kein signifikanter Unterschied zu den anderen Messungen zu erkennen.
Generell wird festgestellt, dass mit zunehmender Anzahl der konkurrierenden Datenströmen die Latenzen leicht steigen. Außerdem wird eine größere Schwankung der Messwerte beobachtet. Bei gleicher Konfiguration werden Unterschiede von mitunter einer μs festgestellt. Somit wird auch deutlich, dass mehrere parallele Ströme sich negativ auf den Jitter auswirken.

Zur Verifizierung der Ergebnisse wird die erste Messung mit der in Abbildung 4.27 beschriebenen Konfiguration wiederholt. Dabei werden zur Belastung des Links nun mehrere Datenströme pro *VL* eingesetzt. Das Ergebnis ist in Abbildung 4.30 dargestellt. Die *rote* Kurve dient als Referenzmessung zur vorherigen Messreihe. Durch die beiden anderen Messungen wird ersichtlich, dass mehrere Ströme pro

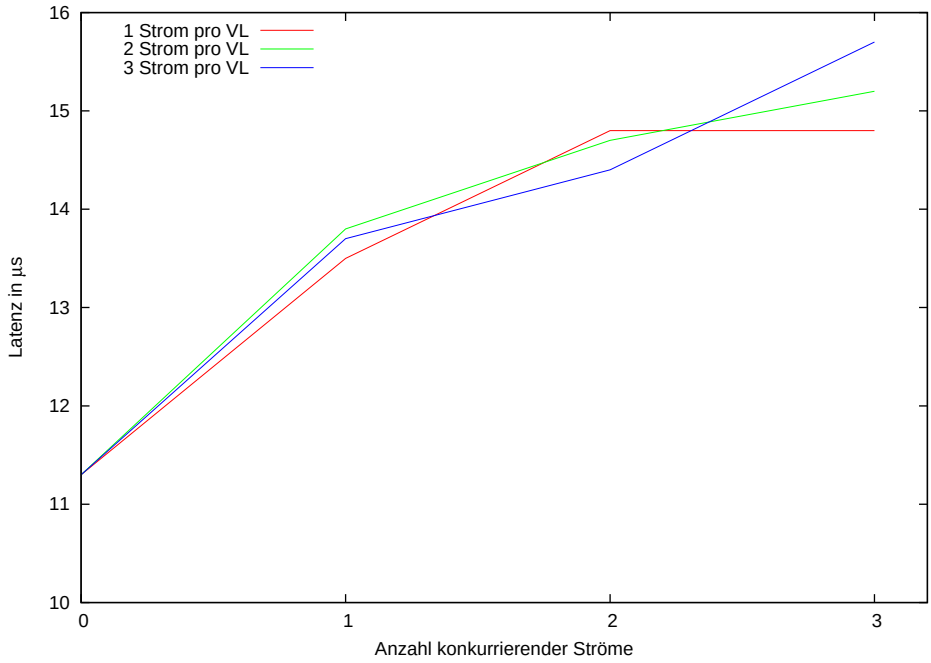

Abbildung 4.30: Latenz bei mehreren Datenströmen pro *VL*

VL kaum Einfluss auf die Latenz haben. Es wird lediglich festgestellt, dass bei vielen parallelen Datenströmen die Latenz im Vergleich leicht steigt.

Aus den bisherigen Messungen bleibt festzuhalten, dass die Latenz einer hoch priorisierten *VL* von anderem hoch priorisierten Verkehr nur wenig beeinflusst wird. Auch durch Mehrfachnennungen einer *VL* innerhalb der hoch priorisierten Liste wird die Latenz, im Gegensatz zu den ursprünglichen Erwartungen, nicht signifikant verändert. Auch bei Latenzmessungen mit größeren Paketen ist, bis auf die grundsätzlich etwas höhere Latenz, kein Unterschied zu den hier detailliert vorgestellten Ergebnissen messbar.

Im Folgenden wird die Latenz bei Verwendung von hoch und niedrig priorisierten Datenströmen untersucht. Dabei wird auch der Einfluss des *High-Limits* betrachtet.

Für die Messungen werden zwei unterschiedlichen Konfigurationen der *VL-Arbitration* vorgenommen. Zunächst wird die in Abbildung 4.31 dargestellten Konfiguration verwendet. Darin werden drei *VLs* mit gleichem Gewicht hoch priori-

```
qos_vlarb_high 1:128,2:128,3:128
qos_vlarb_low 4:8,0:1
```

Abbildung 4.31: Erste Konfiguration des *SM* zur Latenzmessung bei niedrig Priorisierung

siert. Auf der niedrig priorisierten *VL 4* wird die Latenz gemessen. Wie schon gezeigt, spielt der genaue Betrag des Gewichtes keine weitere Rolle. Auch *VL 0* wird niedrig priorisiert, um den Default-Verkehr aufzufangen. Während den Messungen wird *VL 0* allerdings nicht weiter verwendet. Daher wird die Messung nicht beeinflusst. Die zweite Konfiguration ist in Abbildung 4.32 dargestellt. Darin wird die zur Messung verwendete *VL 4* mit geringem Gewicht auch in die

```
qos_vlarb_high 4:8,1:128,4:8,2:128,4:8,3:128
qos_vlarb_low 4:8,0:1
```

Abbildung 4.32: Zweite Konfiguration des *SM* zur Latenzmessung bei niedrig Priorisierung

hoch priorisierte Liste aufgenommen.
Beide Konfigurationen werden jeweils mit einem *High-Limit* von null und 255 durchgeführt. Das Ergebnis ist in Abbildung 4.33 dargestellt. Unabhängig des Wertes des *High-Limits*, führt die erste Konfiguration bei zunehmenden konkurrierenden Datenströmen zu einer stark ansteigenden Latenz. Für ein *High-Limit* von 255 entspricht der Verlauf den Erwartungen. Für einen Wert von null hingegen wird eine konstant niedrige Latenz erwartet, die durch die Messung nicht bestätigt wird. Allerdings deckt sich das Verhalten bei einem Wert von null, mit den bei der Bandbreitenmessungen gewonnenen Erkenntnissen.
Mit der zweiten Konfiguration wird ebenfalls unabhängig des *High-Limits* ein gutes und auch so erwartetes Ergebnis erzielt. Allerdings wird, anders als das Ergebnis zeigt, eine niedrigere Latenz bei einem *High-Limit* von null erwartet.
Zusätzlich zu den beiden vorgestellten Konfigurationen wird in Abbildung 4.33 noch eine dritte betrachtet. Diese ist in Abbildung 4.34 dargestellt. Darin wird nur die *VL 1* zur Latenzmessung hoch priorisiert. Neben *VL 0* mit einem Gewicht von eins, werden die drei anderen *VLs* mit einem Gewicht von 128 niedrig priorisiert. Auch diese Konfiguration liefert zufriedenstellende Ergebnisse bei mehreren konkurrierenden Strömen.

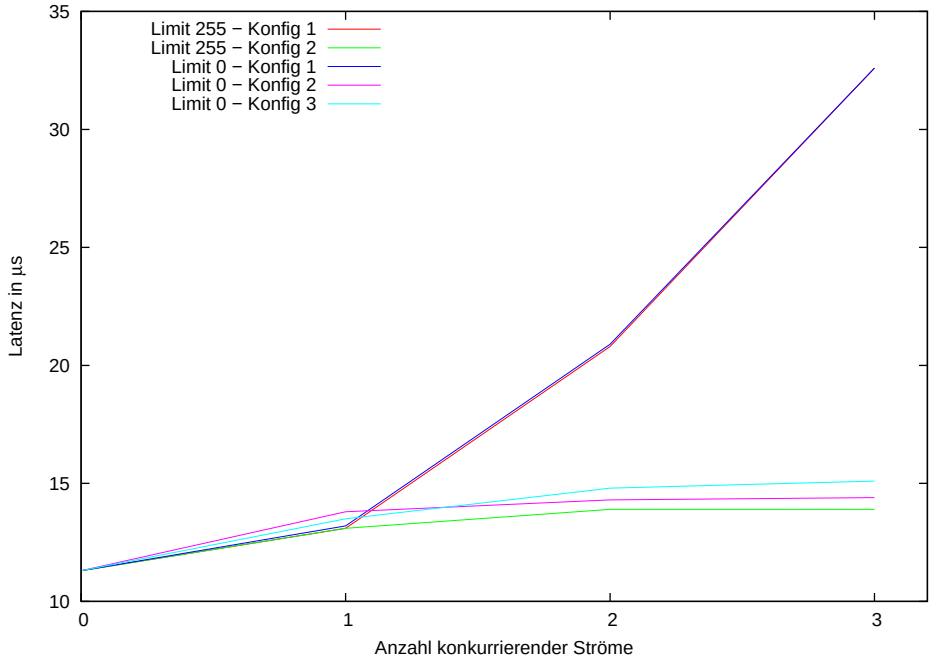

Abbildung 4.33: Auswertung der Latenz bei unterschiedlicher Konfigurationen

```
qos_vlarb_high 4:1
qos_vlarb_low 1:128,2:128,3:128,0:1
```

Abbildung 4.34: Zusätzliche Konfiguration des *SM* zur Latenzmessung

Durch die letzten Messungen wird deutlich, dass bei richtiger Konfiguration sehr gute Latenzen erzielt werden können. Dabei fällt auf, dass die Funktionalität des *High-Limits* wie auch bei den Bandbreitenmessungen nicht dem Standard entspricht. Vor allem durch niedrige Werte des Limits wurde eine geringe Latenz für niedrig priorisierte Datenströme bei gleichzeitigen hoch priorisierten Datenströmen erwartet. Anhand der durchgeführten Messungen wird jedoch deutlich, dass geringe Latenzen nur erreicht werden können, wenn die entsprechende *VL* hoch priorisiert wird. Dies bedeutet, wie auch bei der Bandbreite, eine erhebliche Einschränkung der Funktionalität von *QoS*.

Des Weiteren bleibt festzuhalten, dass durch die Gewichte der *VLs* kein großer Einfluss auf die Latenz der Datenströme in dieser *VL* genommen wird.

4.4.3 Service Level

Gerade für die tatsächliche Umsetzung von *QoS* ist die Zuweisung der *SL* auf die Anwendungen beziehungsweise auf die einzelnen Protokolle von zentraler Bedeutung. Daher werden im Folgenden die möglichen Konfigurationsparameter der *Policies*, sowie die Zuweisung an sich betrachtet. Dafür werden die Abschnitte der *Advanced Syntax* nacheinander analysiert und die noch nicht betrachteten Parameter der *Simple Syntax* betrachtet.

In der *Advanced Syntax* können die *Port Groups* anhand von *GUIDs*, sowie aufgrund von *IPoIB-Partitionen* definiert werden. Daraus wird jeweils eine Liste von Ports erstellt, auf die aus den *Match Rules* zugegriffen werden kann. Die grundlegende Funktionalität einer *Port Group*, die über *GUIDs* definiert wird, wurde bereits in Kapitel 4.3 nachgewiesen (vgl. Abbildung 4.17). Auch durch die Verwendung eines *Partition-Keys* wird die entsprechende Funktionalität erreicht.

Im Abschnitt der *QoS-Level* wird zunächst die Funktionsweise der Parameter zur Begrenzung der *MTU* und der maximalen Senderate im Abschnitt der *QoS Levels* überprüft. Dazu wird die in Abbildung 4.35 dargestellte Konfiguration der *Policies* vorgenommen. Darin wird festgelegt, dass wenn Knoten 7 Daten sendet,

```
port-groups
  port-group
    name: Node7
    port-guid: 0x002590ffff2fb1f5
  end-port-group
end-port-groups

qos-levels
  qos-level
    name: DEFAULT
    sl: 1
  end-qos-level
  qos-level
    name: Level2
    sl: 2
    mtu-limit: 2
    rate-limit: 2
  end-qos-level
end-qos-levels

qos-match-rules
  qos-match-rule
    source: Node7
    qos-level-name: Level2
  end-qos-match-rule
end-qos-match-rules
```

Abbildung 4.35: *Policy* mit Begrenzung der *MTU* und der Senderate

das *QoS-Level Level2* verwendet wird. Dadurch werden die Pakete mit dem *SL 2* markiert, die Senderate auf 2,5 Gb/s beschränkt, sowie die *MTU* auf 512 Bytes

begrenzt. Die Werte für die *MTU* und die Senderate werden in der Konfiguration jeweils über die Zahl zwei gesetzt. Die Umsetzung der Konfigurationszahlen auf die tatsächlichen Werte erfolgt nach einer im Infiniband-Standard definierten Tabelle. Für alle anderen gesendeten Pakete wird der Default angewendet und somit das *SL 1* genutzt.
Die *SL 0-2* werden entsprechend über die *VL 0-2* übertragen. *VL 1* und *2* werden jeweils mit einem Gewicht von 128 hoch priorisiert, *VL 0* mit einem Gewicht von eins niedrig.
Zur Überprüfung wird eine Bandbreitenmessung mittels *iperf* zwischen den Knoten 6 (IP: 192.168.120.17) und 7 (IP: 192.168.120.18) durchgeführt. Parallel dazu werden die versendeten Pakete mittels *ibdump* aufgezeichnet. In Abbildung 4.36 ist ein Paket, dass von Knoten 7 an Knoten 6 geschickt wird, dargestellt. Es wird

```
Frame 975695: 2082 bytes on wire (16656 bits), 2082 bytes captured (16656 bits) on interface 0
Extensible Record Format
  InfiniBand
  Local Route Header
    0010 .... = Virtual Lane: 0x02
    .... 0000 = Link Version: 0
    0010 .... = Service Level: 2
    .... 00.. = Reserved (2 bits): 0
    .... ..10 = Link Next Header: 0x02
    Destination Local ID: 13
    0000 0... .... .... .... = Reserved (5 bits): 0
    .... .010 0000 1000 = Packet Length: 520
    Source Local ID: 15
  Base Transport Header
  DETH - Datagram Extended Transport Header
  IBA Payload - appears to be EtherType encapsulated
  Invariant CRC: 0x241e551d
  Variant CRC: 0x0e1f
  Internet Protocol Version 4, Src: 192.168.120.18 (192.168.120.18), Dst: 192.168.120.17 (192.168.120.17)
  Transmission Control Protocol, Src Port: 58557 (58557), Dst Port: commplex-link (5001), Seq: 22779493, Ack: 1, Len: 2004
  Data (2004 bytes)
```

Abbildung 4.36: Paket der limitierten Messung mit *iperf*

deutlich, dass die grundsätzliche Einteilung in das *QoS Level* über die Gruppe *Node 7* funktioniert, da das richtige *SL 2* beobachtet wird. In Gegenrichtung wird der Default angewendet. Die Pakete werden dementsprechend dem *SL 1* zugeordnet. Allerdings zeigt sich, dass keine Einschränkung der *MTU* vorgenommen wird, da das dargestellte Paket eine Größe von 2082 Bytes aufweist. Die 2082 Bytes ergeben sich einerseits aus den 2004 Bytes Daten, mit je 20 Bytes *IP-* und *TCP-Header*. Dies entspricht zuzüglich des 4 Byte großen *Ether Typs*, der das eingekapselte Protokoll (in diesem Fall *TCP*) angibt, der im Netz global definierten *MTU* von 2048 Bytes. Andererseits werden 28 Byte für drei unterschiedliche Infiniband-Header, sowie 6 Byte für Prüfsummen verwendet, wodurch sich in Summe 2082 Bytes ergeben.
Zur Kontrolle der Senderate wurde die Messung ohne eine Aufzeichnung der Pakete durch *ibdump* wiederholt. Hierbei wurde ebenfalls keine Reduzierung der gemessenen Bandbreite festgestellt.
Als Ergebnis bleibt festzuhalten, dass durch die Parameter *mtu-limit* und *rate-limit* keinen Einfluss auf die Verbindung genommen wird. Somit gilt an dieser Stelle nur die Konfiguration der *VL-Arbitration*.

Im Abschnitt der *Match Rules* werden die Parameter *Source* und *Destination*, sowie *Pkey* und *Service-ID* betrachtet.
Die Funktionalität des Parameters *Source* wurde bereits im vorherigen Test explizit nachgewiesen. Mit einer analogen Konfiguration mit dem Parameter *Destination* wird mit dem selben Versuch ebenfalls das erwartete Ergebnis erzielt. Allerdings greift die Einteilung nur, wenn *ULPs* oder Anwendungen mit einer entsprechenden Implementierung verwendet werden. Kann explizit ein *SL* gesetzt werden, wie zum Beispiel bei *qperf*, so wird immer das explizite *SL*, beziehungsweise der in der Anwendung vorgesehene Default, verwendet. Das Verhalten des Parameters *target-port-guid* der *Simple Syntax* ist zu dem genauer betrachteten Parameter *Destination* im Prinzip analog. In der *Simple Syntax* können nur keine *Port Groups* verwendet werden.
Für die Überprüfung der Zuweisung eines *SL* anhand des *IPoIB-Partition-Keys* wird die *Simple Syntax* genutzt. Dazu wird die in Abbildung 4.37 dargestellte Konfiguration verwendet. Dadurch wird bei einer Kommunikation mit *IPoIB*

```
qos-ulps
  default : 1
  ipoib, pkey 0xfffe : 2
  ipoib: 3
end-qos-ulps
```

Abbildung 4.37: *Policy* mit Unterscheidung von *Partition-Keys*

und einem *Partition-Key* von *0xffffe* das *SL 2* und bei jeglichem anderen *IPoIB-Verkehr* das *SL 3* genutzt.
Zur Überprüfung wird eine zusätzliche Partition mit dem entsprechenden *Partition-Key 0xffffe* erstellt. Darin sind die Knoten 6 und 7 enthalten. Zwischen den beiden Knoten wird eine Bandbreitenmessung mittels *iperf* durchgeführt und die gesendeten Pakete mit *ibdump* aufgezeichnet. Ein aufgezeichnetes Paket ist in Abbildung 4.38 dargestellt. Gesondert hervorgehoben werden das genutzte *SL 2* beziehungsweise die *VL 2*, sowie der gesetzte *Partition-Key* von 65534 (entspricht *0xffffe*). Durch die Verwendung der anderen Partition werden im Vergleich zu den vorherigen Tests andere *IP-Adressen* für die Knoten 6 (192.168.121.17) und 7 (192.168.121.18) verwendet.
Mit dieser Überprüfung wird nachgewiesen, dass eine Zuweisung des *SL* aufgrund der genutzten Partition wie erwartet vorgenommen wird.

Tests zur Zuweisung eines *SL* aufgrund der *Service ID* können nicht durchgeführt werden. Das angegebene Protokoll *SDP* wird in der aktuellen *Mellanox-OFED-Version* nicht mehr unterstützt. Der Nachfolger, das Protokoll *Reliable Datagram Sockets (RDS)*, befindet sich zur Zeit noch im Entwicklungsstadium. In einem ersten Test wird deutlich, dass darin zur Zeit keine Unterstützung von *QoS* vor-

```
Frame 35: 2082 bytes on wire (16656 bits), 2082 bytes captured (16656 bits) on interface 0
Extensible Record Format
InfiniBand
  Local Route Header
    0010 .... = Virtual Lane: 0x02
    .... 0000 = Link Version: 0
    0010 .... = Service Level: 2
    .... 00.. = Reserved (2 bits): 0
    .... ..10 = Link Next Header: 0x02
    Destination Local ID: 13
    0000 0... .... .... = Reserved (5 bits): 0
    .... .010 0000 1000 = Packet Length: 520
    Source Local ID: 15
  Base Transport Header
    Opcode: 100
    0... .... = Solicited Event: False
    .1.. .... = MigReq: True
    ..00 .... = Pad Count: 0
    .... 0000 = Header Version: 0
    Partition Key: 65534
    Reserved (8 bits): 0
    Destination Queue Pair: 0x000300
    0... .... = Acknowledge Request: False
    .000 0000 = Reserved (7 bits): 0
    Packet Sequence Number: 24
  DETH - Datagram Extended Transport Header
  IBA Payload - appears to be EtherType encapsulated
  Invariant CRC: 0xbea27655
  Variant CRC: 0xfa8f
Internet Protocol Version 4, Src: 192.168.121.18 (192.168.121.18), Dst: 192.168.121.17 (192.168.121.17)
Transmission Control Protocol, Src Port: 43791 (43791), Dst Port: commplex-link (5001), Seq: 42109, Ack: 1, Len: 2004
Data (2004 bytes)
```

Abbildung 4.38: Paket der Verbindung mit gesondertem *Partition-Key*

handen ist. Aufgrund der Sonderbehandlung von *IPoIB* über die *Partition-Keys* und von *MPI* kann kein weiteres *ULP* herangezogen werden.

Die Zuweisung des *SL* bei der Verwendung von *MPI* wird direkt über die *MPI-Implementierung* gesteuert. Im Folgenden wird mit *Open MPI* in der Version *1.8.4* gearbeitet. Zudem wird die *Mellanox Messaging Library (MXM)* in Version *3.2.2990-1* verwendet. Sie bietet, durch die effizientere Ausnutzung der Infiniband-Struktur Verbesserungen für Bibliotheken zur parallelen Kommunikation. Unter anderem wird auch die Zuweisung eines *SL* für *Open MPI* ermöglicht. Die Unterstützung von *QoS* in *MXM* befindet sich allerdings noch im Entwicklungsstadium [3].
Für die Zuweisung der *SL* werden in *MXM* zwei Parameter verwendet, über die ein Intervall an möglichen *SLs* definiert wird. Aus diesem Intervall wird für jede *MPI-Verbindungsrichtung* zufällig ein *SL* gewählt. Über den Parameter *mxm_ib_first_sl* wird das zahlenmäßig kleinste *SL*, das verwendet werden kann, angegeben. Über *mxm_ib_num_sls* wird die Größe des Intervalls festgelegt. Wird beispielsweise *mxm_ib_first_sl* auf zwei und *mxm_ib_num_sls* auf drei gesetzt, können die *SL 2-4* verwendet werden.
Zur Überprüfung der Umsetzung wird, wie in Abbildung 4.39 dargestellt, auf Basis von *MPI* eine *Hello-World-Anwendung* auf Knoten 6 gestartet. Alle gesendeten Pakete werden parallel dazu mittels *ibdump* aufgezeichnet. Das Kommando *mpirun* wird mit mehreren Parametern aufgerufen. Über *mtl_mxm_np 0* wird

```
mpirun -mca mtl_mxm_np 0
       -mca mca_base_env_list "MXM_IB_FIRST_SL=3;MXM_IB_NUM_SLS=1"
       --host n007,n006 hello
```

Abbildung 4.39: Starten einer *MPI-Anwendung* mit *Open MPI* unter Verwendung der *MXM-Bibliothek*

die *MXM* für jede Prozessanzahl aktiviert. Des Weiteren wird über die vorgestellten Parameter für jeglichen *MPI-Verkehr* das *SL 3* definiert. Die Anwendung *hello* wird lokal auf Knoten 6, sowie auf Knoten 7 ausgeführt.
In Abbildung 4.40 ist ein Paket aus dem *MPI-Verkehr* dargestellt. Für die Ver-

```
Frame 47: 118 bytes on wire (944 bits), 118 bytes captured (944 bits) on interface 0
Extensible Record Format
InfiniBand
  Local Route Header
    0011 .... = Virtual Lane: 0x03
    .... 0000 = Link Version: 0
    0011 .... = Service Level: 3
    .... 00.. = Reserved (2 bits): 0
    .... ..10 = Link Next Header: 0x02
    Destination Local ID: 15
    0000 0... .... .... .... = Reserved (5 bits): 0
    .... .000 0001 1101 = Packet Length: 29
    Source Local ID: 13
  Base Transport Header
  DETH - Datagram Extended Transport Header
  Invariant CRC: 0xacacf70c
  Variant CRC: 0x50cf
  Data (84 bytes)
```

Abbildung 4.40: *Open MPI-Paket* mit gesetztem *SL* durch die *MXM-Bibliothek*

bindung wird das *SL 3* wie erwartet verwendet. Wird für *MXM* ein Intervall und nicht ein einzelner Wert für die *SLs* angegeben, so wird für jede Verbindungsrichtung ein *SL* zufällig aus dem Intervall ausgewählt. Das heißt, dass für eine logische *MPI-Verbindung* durchaus zwei unterschiedliche *SLs* verwendet werden können. Das Verhalten der *MXM-Bibliothek* wurde zusätzlich mit dem *OSU MPI Bandwidth Test* nachgewiesen.
Es bleibt festzuhalten, dass eine Zuweisung der *SLs* für *Open MPI* durch die *MXM* ermöglicht wird. Durch die Definition eines Intervalls, werden zudem mehrere *SLs* für verschiedene Verbindungen beziehungsweise Verbindungsrichtungen verwendet.

Allgemein kann durch die gegebenen Möglichkeiten zur Zuweisung der *SLs* komplexe Strukturen umgesetzt werden. Allerdings greifen die Konfigurationen zur Begrenzung der Senderate, sowie der *MTU* in der *Advanced Syntax* nicht. Die Einstellungen können aber über die *VL-Arbitration* vorgenommen werden.

4.5 Auswertung und Anwendung von QoS

In diesem Kapitel wird auf Basis der Ergebnisse aus den Abschnitten in Kapitel 4.4 eine Konfiguration für die in Kapitel 4.1 vorgestellten Anwendungsszenarien erarbeitet. Es wird davon ausgegangen, dass im gesamten Netzwerk bis zu acht verschiedene *VLs* genutzt werden können.

Für das Cluster werden zunächst die unterschiedlichen *SLs* den Anwendungen und Protokollen zugeordnet. Als genereller Default wird das *SL 0* verwendet.
In Abschnitt 4.4.1 wurde gezeigt, dass die Bandbreite bei paralleler Verwendung von mehreren *VLs* höher ist, als bei einer einzelnen. Daher werden für zu priorisierenden *MPI-Anwendungen* die *SL 1-3* zugeordnet. Auch die *IP-Anwendungen* sollen in einem separaten *SL* zusammengefasst werden. Hierfür wird das *SL 4* verwendet.
Die Anbindung der *Lustre-Server* wird in zwei Teile getrennt. Den *Control-Servern* wird das *SL 5* und den *Data-Servern* das *SL 6* zugeordnet. Zudem wird eine Gruppe von Knoten ausgewählt, die eine bessere Performance in der Storage-Anbindung erhalten sollen. Hierfür wird das *SL 7* verwendet.
Für die sich daraus ergebende Konfiguration wird die *Advanced Syntax* genutzt, da eine Zuweisung eines *SL* aufgrund des Senders in der *Simple Syntax* nicht möglich ist. Die Konfiguration ist in den folgenden Abbildungen dargestellt.
Zunächst werden die *Port Groups* betrachtet. Sie sind in Abbildung 4.41 dargestellt. Darin werden die *Port Groups* für die beiden unterschiedlichen *Lustre-*

```
port-groups
  port-group
    name: Storage-Control-Server
    port-guid: **alle GUIDs der Lustre-Control-Server**
  end-port-group
  port-group
    name: Storage-Control-Server
    port-guid: **alle GUIDs der Lustre-Data-Server**
  end-port-group
  port-group
    name: Storage-Prio-Group
    port-guid: **Liste der GUIDs der Prio-Knoten**
  end-port-group
end-port-groups
```

Abbildung 4.41: *Policy-Konfiguration* der *Port Groups* eines Clusters mit priorisiertem *MPI-Verkehr*

Server über die *GUIDs* definiert. Die Knotengruppe mit priorisierter Storage-Anbindung wird ebenfalls über die entsprechenden *GUIDs* der Knoten angegeben.
In der folgenden Abbildung 4.42 sind die einzelnen *QoS-Level* aufgeführt. In den jeweiligen Einträgen wird ein sprechender Name, sowie das entsprechende *SL* angegeben. Über die Namen werden die *QoS-Level* in den *Match-Rules* referenziert.

```
qos-levels
  qos-level
    name: Default
    sl: 0
  end-qos-level
  qos-level
    name: IPoIB-Default
    sl: 4
  end-qos-level
  qos-level
    name: Storage-Control
    sl: 5
  end-qos-level
  qos-level
    name: Storage-Data
    sl: 6
  end-qos-level
  qos-level
    name: Storage-Data-Prio
    sl: 7
  end-qos-level
end-qos-levels
```

Abbildung 4.42: *Policy-Konfiguration* der *QoS-Level* des betrachteten Clusters mit priorisiertem *MPI-Verkehr*

Die *Match-Rules* sind in Abbildung 4.43 dargestellt. Bei der Zuweisung des *IPoIB-Defaults* wird der standardmäßig verwendete *Partition-Key* angegeben. Darüber wird jede *IPoIB-Verbindung*, von der keine gesonderte Partition genutzt wird, abgedeckt. Für die *Match-Rules* bezüglich der Storage-Anbindung wird jeweils über den Parameter *Destination* das Routingziel eines Paketes zur Bestimmung des *SL* verwendet. Für die priorisierte Knotengruppe wird zusätzlich der Parameter *Source* genutzt. Es muss darauf geachtet werden, dass die speziellste Regel zuerst aufgeführt wird, da im Entscheidungsfall die Regeln sukzessiv abgearbeitet werden. Außerdem muss bei Verwendung der Parameter *Source* und *Destination* darauf geachtet werden, dass beide Richtungen der Verbindung separat angegeben werden müssen. Ansonsten würden auch nur die Pakete einer Richtung mit dem entsprechenden *SL* markiert werden.
Die *SLs* für *MPI-Anwendungen* werden in der Konfiguration nicht aufgeführt. Sie müssen, wie in Abschnitt 4.4.3 beschrieben, bei Aufruf von *mpirun* explizit gesetzt werden. Alternativ können die Defaults der Parameter in einer Konfigurationsdatei angepasst werden. Zur gleichmäßigen Verteilung des Verkehrs auf die drei *SLs* wird ein *SL-Intervall* definiert. Dadurch werden die drei *SLs* automatisch verwendet.
Durch die vorgestellte Zuweisung der *SLs* werden die unterschiedlichen Datenströme voneinander getrennt. Somit kann eine Priorisierung über das *SL to VL Mapping*, sowie die Gewichtung der *VLs* vorgenommen werden.

Damit alle *SLs* getrennt behandelt werden können, wird jedem *SL* eine separate *VL* zugeordnet. Der Einfachheit halber werden die *SLs* auf die numerisch ent-

```
qos-match-rules
  qos-match-rule
    pkey: 0xffff
    qos-level-name: IPoIB-Default
  end-qos-match-rule
  qos-match-rule
    source: Storage-Prio-Group
    destination: Storage-Data-Server
    qos-level-name: Storage-Data-Prio
  end-qos-match-rule
  qos-match-rule
    source: Storage-Data-Server
    destination: Storage-Prio-Group
    qos-level-name: Storage-Data-Prio
  end-qos-match-rule
  qos-match-rule
    destination: Storage-Control-Server
    qos-level-name: Storage-Control
  end-qos-match-rule
      qos-match-rule
    source: Storage-Control-Server
    qos-level-name: Storage-Control
  end-qos-match-rule
  qos-match-rule
    destination: Storage-Data-Server
    qos-level-name: Storage-Data
  end-qos-match-rule
  qos-match-rule
    source: Storage-Data-Server
    qos-level-name: Storage-Data
  end-qos-match-rule
end-qos-match-rules
```

Abbildung 4.43: *Policy-Konfiguration* der *Match-Rules* des betrachteten Clusters mit priorisiertem *MPI-Verkehr*

sprechenden *VLs* umgesetzt. Alle anderen *SLs* werden *VL 15* zugeordnet und dadurch blockiert. Über die Gewichtung beziehungsweise Zuordnung der einzelnen *VLs* in die beiden Listen wird schließlich die Priorität festgelegt.
Das *SL to VL Mapping*, sowie die *VL Arbitration* ist in Abbildung 4.44 dargestellt. Für die Konfiguration werden alle acht *VLs* benötigt. Damit für die *Lustre-*

```
qos_max_vls 8
qos_high_limit 0
qos_vlarb_high 5:1
qos_vlarb_low 0:4,1:64,2:64,3:64,4:8,5:0,6:32,7:64
qos_sl2vl 0,1,2,3,4,5,6,7,15,15,15,15,15,15,15,15
```

Abbildung 4.44: *SM-Konfiguration* des betrachteten Clusters mit priorisiertem *MPI-Verkehr*

Control-Server eine konstant niedrige Latenz garantiert werden kann, wird nur *VL 5* mit einem Gewicht von eins hoch priorisiert und ein *High-Limit* von null

definiert. Dadurch wird nach den gewonnenen Erkenntnissen neben der geringen Latenz erreicht, dass durch den restlichen Verkehr aus der niedrig priorisierten Liste die volle Bandbreite genutzt werden kann. Zudem ist für den Verkehr zu den *Lustre-Control-Servern* ein Gewicht von eins ausreichend, da nur kleine Datenmengen verschickt werden. Die anderen *VLs* werden so gewichtet, dass wenn alle *SLs* voll belastet werden, 64 % der Bandbreite von den *MPI-Anwendungen* genutzt wird. Von den Verbindungen zum Storage werden währenddessen insgesamt 32 % der Bandbreite beansprucht. Für die *IPoIB-Anwendungen* werden im Überlastfall nur 2,7 % und für den Default 1,3 % der Bandbreite zur Verfügung gestellt.
Die Ergebnisse einer Kontrollmessung mit der vorgestellten Konfiguration sind in Abbildung 4.45 dargestellt. Für die Messung werden zeitgleich Bandbreitenmessungen auf den *SL 0-4* und den *SL 6-7*, sowie eine Latenzmessung auf *SL 5* zwischen den Knoten 6 und 7 mittels *qperf* durchgeführt. Die gemessenen Werte

```
Default: 46.3 MB/s
IPoIB: 92.6 MB/s
MPI: 2223 MB/s
Storage-Prio: 741 MB/s
Storage: 371 MB/s

Storage-Control: 15.1 us
```

Abbildung 4.45: Ergebnisse der Kontrollmessung für das betrachtete Cluster

für die einzelnen Verbindungsarten entsprechen dabei genau den zuvor nach der Konfiguration ermittelten Prozentsätzen. Auch durch die zeitgleich durchgeführte Latenzmessung wird wie erwartet ein geringer Wert von 15.1 μs ermittelt. Somit wird die volle Funktionalität der erarbeiteten Konfiguration bestätigt. Die korrekte Funktionsweise der Zuweisung der *SLs* wurde bereits in Abschnitt 4.4.3 nachgewiesen.

Im Normalfall wird davon ausgegangen, dass nur *MPI-Anwendungen*, sowie nicht priorisierte Storage-Verbindungen genutzt werden. Das bedeutet, dass dann durch die *MPI-Anwendungen* bis zu 86 % der Bandbreite genutzt werden können. Die restlichen 14 % werden durch die Verbindungen zum Storage verwendet.

Durch die Konfiguration wird erreicht, dass der *MPI-Verkehr* bevorzugt vor der Storage-Anbindung behandelt wird. Durch andere Anwendungen, wie zum Beispiel *IPoIB-Anwendungen*, kann kaum noch Bandbreite beansprucht werden. Dadurch wird dem Ziel entsprechend die Performance des Clusters hinsichtlich der Netzwerkleistung für *MPI-Anwendungen* verbessert. Zudem wird bei datenintensiver Kommunikation die Verwendung von *MPI* gegenüber *IPoIB* auf Nutzerseite gefördert.

Für die Umsetzung von *QoS* in dem in Kapitel 4.1 vorgestellten Datacenter werden zunächst die *SLs* definiert. Als genereller Default wird wie auch zuvor das *SL 0* verwendet.
Damit eine separate Behandlung für die Workstations, sowie für die zu priorisierenden *Fat Nodes* innerhalb des Cloud-Servers vorgenommen werden kann, müssen hierfür unterschiedliche *SLs* definiert werden. Für die Verbindung zwischen den Workstations und dem Storage-System wird das *SL 1* und für die Verbindung zum Datenbank-System das *SL 2* verwendet. Analog wird für die Verbindungen zwischen den *Fat-Nodes* und dem Storage das *SL 3* und zum Datenbank-System das *SL 4* genutzt.
Für alle Verbindungen des Backup-Systems wird unabhängig des Ausgangs- beziehungsweise des Zielsystems das *SL 5* definiert. Eine weitere Unterscheidung wird nicht vorgenommen, da die genutzte Bandbreite des gesamten Systems eingeschränkt werden soll. Konfigurationen bezüglich des Ablaufs der Backups der einzelnen Systeme können direkt im Backup-System realisiert werden.
Für die Umsetzung der Konfiguration wir die *Advanced Syntax* verwendet. Wie auch bei der Konfiguration des Clusters ist die Verwendung der *Simple Syntax* nicht möglich, da auf den Parameter *Source* zurückgegriffen wird. Die definierten *Port Groups* sind in Abbildung 4.46 dargestellt. Darin werden die einzelnen

```
port-groups
  port-group
    name: Storage-Server
    port-guid: **alle GUIDs der Storage-Server**
  end-port-group
  port-group
    name: Datenbank-Server
    port-guid: **alle GUIDs der DB-Server**
  end-port-group
  port-group
    name: Backup-Server
    port-guid: **Liste der GUIDs Backup-Server**
  end-port-group
  port-group
    name: VM-Server-WS
    port-guid: **Liste der GUIDs der Workstation-VMs**
  end-port-group
  port-group
    name: VM-Server-FN
    port-guid: **Liste der GUIDs der Fat Nodes**
  end-port-group
end-port-groups
```

Abbildung 4.46: *Policy-Konfiguration* der *Match-Rules* des betrachteten Datacenters

Server-Systeme jeweils über die entsprechenden *GUIDs* referenziert. Die *QoS-Level* sind in Abbildung 4.47 dargestellt. Darin werden, wie auch bei der Konfiguration des Clusters, die einzelnen *SLs* mit sprechenden Namen versehen, sodass sie leicht referenziert werden können.

```
qos-levels
  qos-level
    name: Default
    sl: 0
  end-qos-level
  qos-level
    name: WS-Storage
    sl: 1
  end-qos-level
  qos-level
    name: WS-DB
    sl: 2
  end-qos-level
  qos-level
    name: FN-Storage
    sl: 3
  end-qos-level
  qos-level
    name: FN-DB
    sl: 4
  end-qos-level
  qos-level
    name: Backup
    sl: 5
  end-qos-level
end-qos-levels
```

Abbildung 4.47: *Policy-Konfiguration* der *QoS-Level* eines Datacenters

Schließlich werden die *Match-Rules* definiert. Sie sind in Abbildung 4.48 dargestellt. Durch jede *Match-Rule* wird eine Verbindungsrichtung zwischen den Systemen mittels der Parameter *Source* und *Destination* beschrieben. Beide Richtungen einer logischen Verbindung werden dem selben *QoS-Level* zugeordnet. Dadurch werden alle Pakete einer Verbindung mit dem gleichen *SL* versehen.
Durch die Aufteilung des *VM-Servers* müssen für beide Bereiche separate *Match-Rules* erstellt werden, durch die auf andere *QoS-Level* verwiesen wird.
Die Verbindungen zum Backup-Server werden nur über den Parameter *Destination*, beziehungsweise vom Backup-Server aus über den Parameter *Source* beschrieben. Dadurch wird über die beiden *Match-Rules* der gesamte Backup-Verkehr unabhängig des Zielsystems mit dem entsprechenden *QoS-Level* verknüpft.
Durch diese Konfiguration wird erreicht, dass jeder Verbindung, je nach Kommunikationspartnern, ein separates *SL* zugeordnet wird.

Das *SL to VL Mapping* wird äquivalent zum vorgestellten Cluster vorgenommen. Das heißt, dass allen *SLs* eine eigene *VL* zugewiesen wird und die Zuweisung numerisch entsprechend erfolgt. Die nicht genutzten *SL 6-15* werden über *VL 15* blockiert.
Die eigentliche Priorisierung der *SLs* wird über die *VL Arbitration* vorgenommen. Die entsprechende Konfiguration ist in Abbildung 4.49 dargestellt. Generell werden maximal sechs verschiedene *VLs* verwendet. Für die Verbindung zu den

```
qos-match-rules
  qos-match-rule
    source: VM-Server-WS
    destination: Storage-Server
    qos-level-name: WS-Storage
  end-qos-match-rule
  qos-match-rule
    source: Storage-Server
    destination: VM-Server-WS
    qos-level-name: WS-Storage
  end-qos-match-rule
  qos-match-rule
    source: VM-Server-WS
    destination: Datenbank-Server
    qos-level-name: WS-DB
  end-qos-match-rule
  qos-match-rule
    source: Datenbank-Server
    destination: VM-Server-WS
    qos-level-name: WS-DB
  end-qos-match-rule
  qos-match-rule
    source: VM-Server-FN
    destination: Storage-Server
    qos-level-name: FN-Storage
  qos-match-rule
  end-qos-match-rule
    source: Storage-Server
    destination: VM-Server-FN
    qos-level-name: FN-Storage
  end-qos-match-rule
  qos-match-rule
    source: VM-Server-FN
    destination: Datenbank-Server
    qos-level-name: FN-DB
  qos-match-rule
  end-qos-match-rule
    source: Datenbank-Server
    destination: VM-Server-FN
    qos-level-name: FN-DB
  end-qos-match-rule
  qos-match-rule
    destination: Backup-Server
    qos-level-name: Backup
  end-qos-match-rule
  qos-match-rule
    source: Backup-Server
    qos-level-name: Backup
  end-qos-match-rule
end-qos-match-rules
```

Abbildung 4.48: *Policy-Konfiguration* der *Match-Rules* des betrachteten Datacenters

Workstations wird eine sehr geringe Latenz benötigt. Durch die über ein Gateway ans Netz angeschlossen Workstation-Clients wird der Default und somit das *SL 0* verwendet. Daher wird das *SL 0* als einziges hoch priorisiert und ein *High-Limit* von null definiert. Über diese Verbindungen werden nur wenige Daten versendet,

```
qos_max_vls 6
qos_high_limit 0
qos_vlarb_high 0:1
qos_vlarb_low 1:16,2:16,3:128,4:128,1:16,2:16,5:35
qos_sl2vl 0,1,2,3,4,5,15,15,15,15,15,15,15,15,15,15
```

Abbildung 4.49: *SM-Konfiguration* des betrachteten Datacenters

sodass ein Gewicht von eins ausreichend ist. Alle anderen *VLs* werden so gewichtet, dass bei voller Belastung aller *VLs* maximal 10 % der Bandbreite durch das Backup-System verwendet werden können. Währenddessen kann von den *Fat Nodes* für die Storage- und die Datenanbindung jeweils 36 % der Bandbreite genutzt werden. Für die entsprechenden Verbindungen wird den Workstations jeweils 9 % der Bandbreite zugeordnet. Zur Verbesserung der Latenz der Workstations zum Datenbank- und zum Storage-System wird ihr jeweiliges Gewicht in zwei Einträge aufgeteilt.
Auch für diese Konfiguration wird zwischen den Knoten 6 und 7 eine Kontrollmessung mittels *qperf* erstellt. Dazu werden auf den *SL 1-5* Bandbreitenmessungen und auf *SL 0* eine Latenzmessung durchgeführt. Die Ergebnisse dazu sind in Abbildung 4.50 dargestellt. Auch bei dieser Konfiguration werden die konfigu-

```
Workstation-Storage: 313 MB/s
Workstation-DB: 313 MB/s
Fat Nodes-Storage: 1250 MB/s
Fat Nodes-DB: 1250 MB/s
Backup: 342 MB/s

Workstation-Clients: 15.1 us
```

Abbildung 4.50: Ergebnisse der Kontrollmessung für das betrachtete Datacenter

rierten Prozentsätze der Bandbreite, sowie ein niedrige Latenz für die Anbindung der Workstation-Clients erreicht. Die korrekte Funktionsweise der Zuweisung der *SLs* wurde bereits in Abschnitt 4.4.3 nachgewiesen. Somit wird auch die Funktionalität der für das betrachtete Datacenter erarbeiteten Konfiguration bestätigt.

Durch diese Konfiguration wird die Performance, den in Kapitel 4.1 formulierten Anforderungen entsprechend, gesteigert. Mittels der *Fat Nodes* kann ein Großteil der Performance des Netzes eingenommen werden. Währenddessen kann auf den Workstations mit reduzierter Performance gearbeitet werden. Des Weiteren wird dem Backup-System ausreichend Leistung zur Verfügung gestellt.

5 Zusammenfassung und Ausblick

Durch die in dieser Arbeit gewonnenen Erkenntnisse, wird die Möglichkeit eines Einsatzes von *QoS* auf Infiniband im *Jülich Supercomputing Centre* geschaffen.
Auf Basis der *QoS-Theorie* für das Internet, wurde die Umsetzung von *QoS* auf Infiniband betrachtet und die Auswirkungen der einzelnen Konfigurationsparameter auf Bandbreite, Latenz, sowie die Zuweisung der *SLs* analysiert. Daraus konnten fundierte Erkenntnisse über das Verhalten eines Systems unter Einsatz von *QoS* auf Infiniband gewonnen werden. Mit diesen Erkenntnissen wurden für zwei typische Anwendungsszenarien Konfigurationen entwickelt.
Für das vorgestellte Cluster wurde vor allem eine Priorisierung des *MPI-Verkehrs* vorgenommen. Die anderen Dienste wurden ihren Anforderungen entsprechend in der Konfiguration berücksichtigt.
Im zweiten Anwendungsszenario wurde ein Datacenter betrachtet. Darin wurden Dienste unterschiedlicher Art bezüglich ihrer Anforderungen an das Netz analysiert. Es wurde eine Konfiguration erarbeitet, durch die einem Teil der auf die Dienste zugreifenden Knoten, eine bessere Performance geboten wird. Durch die anderen Knoten können die Dienste dennoch den Anforderungen entsprechend genutzt werden.
Unabhängig der Szenarien können durch die Konfiguration von *QoS* bestimmte Anwendungen bevorzugt werden. Dadurch kann beispielsweise auch speziell auf die Hauptnutzer eines Systems eingegangen werden und die Konfiguration ihren Bedürfnissen entsprechend gestaltet werden. Allerdings ist es auch möglich, dass Nutzer über Priorisierungen, in der Wahl der genutzten Dienste beziehungsweise Software, beeinflusst werden.
Generell wurde gezeigt, dass durch den Einsatz von *QoS* eine Vielzahl neuer Konfigurationsmöglichkeiten entstehen. Somit wird der administrative Einfluss auf die Zusammensetzung des Verkehrs im Netz erhöht. Des Weiteren wurde nachgewiesen, dass durch die vorgenommenen Konfigurationen die Gesamtperformance des Netzes in keinem Fall verschlechtert wird.

Zur leichteren Administration von *QoS* auf Infiniband wäre ein zusätzliches Werkzeug, zur besseren Überprüfung der aktuell vorgenommenen Konfiguration, sinnvoll. Neben der bestehenden Abfragemöglichkeit des *SL to VL Mappings* und der *VL Arbitration* kann bisher das gesetzte *High-Limit* nicht überprüft werden. Des

Weiteren wird ein Kontrollmechanismus für die *Policy-Konfiguration* benötigt, sodass die aktuell umgesetzte Konfiguration verifiziert werden kann.
Zudem könnten die genannten Kontrollen in einen regelmäßigen Test zur Bestätigung der korrekten Funktionsweise des Netzes integriert werden.

Außerdem könnte für Cluster ein Premium-Dienst, durch den Nutzern besonders hohe Performance im Netz garantiert wird, eingeführt werden. Bei Nutzung des Premium-Dienstes, über die Angabe eines Parameters im Batch-System, kann dies in der Abrechnung der verwendeten Ressourcen mit einem eigenen Faktor berücksichtigt werden. Das heißt, dass Nutzern gegen Bezahlung eine besser Performance ihres Jobs garantiert wird.
Auch in einem Datacenter kann ein ähnlicher Premiumdienst bei Zugriff auf Dienste, beispielsweise ein Datenbank-System, erzeugt werden. Die Berechtigung zur Nutzung könnte zum Beispiel über separate *Service-IDs* oder auch durch Zuweisung spezieller Systeme gesteuert werden.

Abschließend bleibt festzuhalten, dass die dargestellten Vorteile, sowie die vielversprechenden weiterführenden Einsatzmöglichkeiten von *QoS* einen Einsatz im Rahmen des Produktionsbetriebes eines Infiniband-Netzwerkes in Erwägung gezogen werden sollte.

Literaturverzeichnis

[1] *One-way transmission time.* Technischer Bericht G.114, International Telecommunication Union, 2003.

[2] *Cisco Visual Networking Index: Forecast and Methodology, 2013-1018.* Technischer Bericht, Cisco Systems, 2014.

[3] *Mellanox Messaging Library User Manual, Rev. 3.0.* Technischer Bericht, Mellanox Technologies, 2014.

[4] *Infiniband Architecture-Specification Volume 1, Release 1.3.* Technischer Bericht, Infiniband Trade Association, 2015.

[5] *Mellanox OFED for Linux User Manual Rev. 2.4-1.00.* Technischer Bericht, 2015.

[6] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang und W. Weiss: *An Architecture for Differentiated Services.* RFC 2475 (Informational), Dezember 1998. `http://www.ietf.org/rfc/rfc2475.txt`.

[7] D. Borman, B. Braden, V. Jacobson und R. Scheffenegger: *TCP Extensions for High Performance.* RFC 7323 (Proposed Standard), September 2014. `http://www.ietf.org/rfc/rfc7323.txt`.

[8] R. Braden, D. Clark und S. Shenker: *Integrated Services in the Internet Architecture: an Overview.* RFC 1633 (Informational), Juni 1994. `http://www.ietf.org/rfc/rfc1633.txt`.

[9] R. Braden, L. Zhang, S. Berson, S. Herzog und S. Jamin: *Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification.* RFC 2205 (Proposed Standard), September 1997. `http://www.ietf.org/rfc/rfc2205.txt`.

[10] D. Crupnicoff, S. Das und E. Zahavi: *Deploying Quality of Service and Congestion Control in Infiniband-based Data Center Networks.* Technischer Bericht, Mellanox Technologies, 2005.

[11] B. Davie, A. Charny, J.C.R. Bennet, K. Benson, J.Y. Le Boudec, W. Courtney, S. Davari, V. Firoiu und D. Stiliadis: *An Expedited Forwarding PHB (Per-Hop Behavior).* RFC 3246 (Proposed Standard), März 2002. `http://www.ietf.org/rfc/rfc3246.txt`.

[12] P. Grun: *Introduction to InfiniBand for End Users.* Technischer Bericht, Mellanox Technologies, 2010.

[13] J. Heinanen, F. Baker, W. Weiss und J. Wroclawski: *Assured Forwarding PHB Group.* RFC 2597 (Proposed Standard), Juni 1999. `http://www.ietf.org/rfc/rfc2597.txt`.

[14] J. Kurose und K. Ross: *Computernetzwerke - Der Top-Down Ansatz.* Pearson, München, 5. Auflage, 2012.

[15] J. Le Boudec und P. Thiran: *Network Calculus - A Theory of Deterministic Queuing Systems for the Internet.* Springer, Berlin, 4. Auflage, 2012.

[16] K. Nichols, S. Blake, F. Baker und D. Black: *Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers.* RFC 2474 (Proposed Standard), Dezember 1998. `http://www.ietf.org/rfc/rfc2474.txt`.

[17] K. Nichols, V. Jacobson und L. Zhang: *A Two-bit Differentiated Services Architecture for the Internet.* RFC 2638 (Informational), Juli 1999. `http://www.ietf.org/rfc/rfc2638.txt`.

[18] T. Shanley: *InfiniBand Network Architecture.* Addison-Wesley, Boston, MA, 4. Auflage, 2007.

[19] S. Shenker, C. Partridge und R. Guerin: *Specification of Guaranteed Quality of Service.* RFC 2212 (Proposed Standard), September 1997. `http://www.ietf.org/rfc/rfc2212.txt`.

[20] A. Tanenbaum und D. Wetherall: *Computernetzwerke.* Pearson, München, 5. Auflage, 2012.

[21] J. Wroclawski: *The Use of RSVP with IETF Integrated Services.* RFC 2210 (Proposed Standard), September 1997. `http://www.ietf.org/rfc/rfc2210.txt`.

Abbildungsverzeichnis